Modelling Socio-Technical Aspects of Organisational Security

Ivanova, Marieta Georgieva

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

Modelling Socio-Technical Aspects of Organisational Security

Marieta G. Ivanova

Kongens Lyngby 2016
PHD-2016-406

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk PHD-2016-406

Summary

Identification of threats to organisations and risk assessment often takes into consideration the pure technical aspects, overlooking the vulnerabilities originating from attacks on a social level, for example social engineering, and abstracting away the physical infrastructure. However, attacks on organisations are far from being purely technical. After all, organisations consist of employees. Often the human factor appears to be the weakest point in the security of organisations. It may be easier to break through a system using a social engineering attack rather than a pure technological one. The StuxNet attack is only one of the many examples showing that vulnerabilities of organisations are increasingly exploited on different levels including the human factor. There is an urgent need for integration between the technical and social aspects of systems in assessing their security. Such an integration would close this gap, however, it would also result in complicating the formal treatment and automatic identification of attacks.

This dissertation shows that applying a system modelling approach to socio-technical systems can be used for identifying attacks on organisations, which exploit various levels of the vulnerabilities of the systems. In support of this claim we present a modelling framework, which combines many features. Based on a graph, the framework presents the physical infrastructure of an organisation, where actors and data are modelled as nodes in this graph. Based on the semantics of the underlying process calculus, we develop a formal analytical approach that generates attack trees from the model.

The overall goal of the framework is to predict, prioritise and minimise the vulnerabilities in organisations by prohibiting the overall attack or at least in-

creasing the difficulty and cost of fulfilling it. We validate our approach using scenarios from IPTV and Cloud Infrastructure case studies.

Resumé

Identifikation af trusler mod organisationer samt risikoanalyse tager ofte kun højde for de rent tekniske aspekter og overser derved svagheder, der stammer fra angreb på et socialt niveau, f.eks. social engineering, og som abstraherer den fysiske infrastruktur væk. Imidlertid er angreb på organisationer langt fra udelukkende af teknisk karakter. Ofte viser det sig, at den menneskelige faktor er det svageste punkt i organisationers sikkerhed. Det kan være lettere at bryde igennem et system ved hjælp af et social engineering-angreb i stedet for med et rent teknisk angreb. StuxNet-angrebet er et af mange eksempler, der viser, at svagheder i organisationer i stigende grad bliver udnyttet på flere niveauer, herunder udnyttelse af den menneskelige faktor. Der er et presserende behov for integration mellem de tekniske og sociale aspekter af systemer, når deres sikkerhed bliver vurderet. En sådan integration ville lukke dette gab, men ville også komplicere den formelle behandling og automatiske identifikation af angreb.

Denne afhandling viser, at en systemmodellerings-fremgang anvendt på socio-tekniske systemer kan bruges til at identificere angreb på organisationer, som udnytter forskellige niveauer af svagheder i systemerne. For at understøtte denne påstand præsenterer vi et modellerings-framework, som kombinerer mange funktioner. Baseret på en graf præsenterer frameworket den fysiske infrastruktur af en organisation, hvor aktører og data er modelleret som knuder i grafen. Baseret på semantikken af den underliggende proces-kalkyle udarbejder vi en formel analytisk tilgang, som genererer angrebstræer ud fra modellen.

Det overordnede mål for frameworket er at forudse, prioritere og minimere svaghederne i organisationer ved at forhindre det overordnede angreb, eller ved at forøge sværhedsgraden og omkostningerne ved at udføre angrebet. Vi validerer vores tilgang ved at bruge scenarier fra casestudier af IPTV og Cloud-infrastruktur.

Preface

This thesis was prepared at DTU Compute, the Department of Applied Mathematics and Computer Science, in the Technical University of Denmark in partial fulfilment of the requirements for acquiring the Ph.D. degree in Computer Science.

The Ph.D. study was carried out under the supervision of Professor Christian W. Probst in the period from January 2013 to January 2016. The research in this thesis was conducted as part of the TRE$_S$PASS project.

A substantial part of the scientific work reported in this thesis is based on joint work with my supervisor, collaborations within the framework of TRE$_S$PASS as well as a collaboration fostered by my supervisor with Florian Kammüller (at that time affiliated with the Middlesex University, London, United Kingdom).

Kongens Lyngby, 14 January 2016
Marieta G. Ivanova

Acknowledgements

First, I would like to thank Marian, a former secretary at Cognitive Systems section at DTU, where I wrote my Master's thesis. Without her I would probably have never met my (at that time potential future) PhD supervisor and respectively apply for the PhD position.

I should like to express my appreciation of my supervisor Christian W. Probst for giving me the chance to work with him, for not only guiding me throughout my studies, but also being an example and a source of inspiration.

I am grateful to Laurie Clarke to host me at her group at Massachusetts University, Amherst, for the first part of my research stay; to Matt Bishop and Sean Peisert for supervising me during the research stay at University of California, Davis. It was a great pleasure to work with both groups.

I would like to thank all members of Language-based technologies (recently renamed to Formal Methods for Safe and Secure Systems), as well as visiting students, who I have had the pleasure to meet during the 3 years, for contributing in their own way to the atmosphere in the section.

Thanks go to Zaruhi, Alessandro and Roberto for being not only good colleagues sharing an office with, but also good friends. Special thanks to Roberto for combining his dynamic life (including his Christmas "hygge") with reading my drafts, providing insightful comments, and making me believe in the moments when I was getting lost in my doubts whether I can complete this dissertation.

Special thanks go to Henrik, for he not only proof-read my entire thesis, but

went out of his way to handle me and my intensive days, especially at the end of the PhD. For he has always been there for me when I needed it.

I would like to express my sincere gratitude to my family and friends, especially my parents, who have constantly been providing me with their unconditional love and support throughout my studies in Denmark, and my entire life in general, without who I would not have made it.

Finally, I would like to thank: my arms, for always being by my side; my legs for always supporting me; and my fingers...because I can always count on them.

Contents

Chapter 1

Introduction

Organisations are a big, quickly changing mixture of technical and social parts - socio-technical systems. Such systems can be attacked on different levels, combining attacks from both the technical and the social part. While the technical aspect of security is well understood, the social part is only partially so. The real problem, however, is the fact that the combination of these two is yet to be understood. Because of this it is hard to identify attacks exploiting the different levels of organisations' vulnerabilities.

In this thesis we show that *applying a formal modelling approach to the study of socio-technical systems allows one to develop algorithms for the automated identification of complex attacks that exploit the interplay between technological and social vulnerabilities.*

1.1 Challenge

Organisations need employees - they are the main driving force of contribution to the success and the development of the organisations. At the same time, the human factor poses difficult problems for organisations. By their very nature, employees have access to and knowledge about the organisation's infrastructure, data, and work-flows, and they use these in their everyday work to fulfil tasks.

Determining whether a certain action by an employee is legitimate (regular work) or illegitimate (insider attack), is close to impossible because the insider's purpose for performing the action is not observable. This problem exists both if the action involves assets that the employee is not allowed to access, and even more so, if the access to the asset is allowed.

Organisations often distinguish between threats originating from the inside and the outside. While this in principle makes sense (and many organisations are much better prepared against attacks from the outside than against those from the inside), it is also risky. If an outsider is able to get an insider to perform a certain action, the insider becomes, voluntarily or involuntarily, part of the attack. Social engineering is a typical kind of attack used in these scenarios [MS03]; it aims at making an insider perform an action that he is allowed to perform, were it part of his daily work.

For various reasons, neither regulating insider actions nor surveilling them are viable options. Over-regulation easily results in disgruntled employees, and the human mind can be ingenious when having to circumvent security precautions and policies. Over-surveillance, on the other hand, is illegal in many parts of the world, and even if it is admissible, it is unclear how to draw meaningful conclusions from huge amounts of the logged or observed data. The risk of both false positives and false negatives easily becomes too high.

What seems like a viable option is to analyse an organisation's vulnerability considering the human factor before an attack, or to use tool support after an attack to narrow down which actions might have occurred as part of the attack [PH08]. Traditional and well-established risk assessment methods can often identify these potential threats, but due to a technical focus, these approaches often abstract away the internal structure of an organisation and ignore human factors when modelling and assessing attacks. To support the threat analysis of organisations, several system models have been introduced that model organisations' infrastructure and actors. Examples for such models include ExASyM [PH08], Portunes [DPH10a], and ANKH [Pie11a]. All these models follow similar ideas, namely the modelling of infrastructure and data, and analysing the modelled organisation for possible attacks.

The system models mentioned above have an important shortcoming: the attacker model and the attacker behaviour are tightly integrated into the system model and the analysis. This is an undesirable property; it means that experimenting with different behaviours and types of insiders is close to impossible [Col09]. It is exactly this tight integration that hampers the models' applicability to analyses of an organisation's vulnerability to different kinds of attacks. It is also difficult, if not impossible, to analyse the effect of changed employee behaviour.

1.2 Thesis Contribution

The core contribution of this thesis is to show that *applying a formal modelling approach to the study of socio-technical systems allows one to develop algorithms for the automated identification of complex attacks that exploit the interplay between technological and social vulnerabilities.*

In order to address the challenge of identifying attacks that exploit organisational vulnerabilities on all different levels of a socio-technical system, namely physical infrastructure, human actors, policies, and processes, we develop a modelling framework where all these relevant aspects of socio-technical systems coexist seamlessly.

We explore different approaches to tackle the problem of defining human behaviour. One approach is to formally model human behaviour as an independent component in the system model. Doing so we aim at enabling behaviour-based analysis which would result in more flexibility compared to existing studies. Due to the irrationality and unpredictability of humans, this proves to be practically impossible. Instead, we try to apply fundamental sociological methods to explain human behaviour using higher order logic. While this approach can contribute to the validation of the attack generation, it still does not properly encompass the irrationality of people.

Our approach models the organisation under scrutiny using a process calculus from the Klaim family [DFP98b; GP03; PHN07a]. This model contains all relevant aspects of a socio-technical system. It also specifies access control policies and trust relations. When evaluating our techniques, we also discuss the apparent problem of obtaining precise models of these properties.

Once the organisation has been modelled, the algorithm based on policy invalidation we propose identifies ways to break a policy in this model. The policy to invalidate can be specified as part of the model, or we try to invalidate all policies in the model. The former approach results in a relatively targeted set of attacks, while the latter, though exhaustive, may contain many attacks that are not of interest.

The attacks discovered by our policy invalidation algorithm are represented in the form of an attack tree. Attack trees [Sch99; KPS14] are widely used by various security analysis techniques; they support an easily accessible tree-like structure that can be visualised and understood by non-experts. At the same time, they can be subjected to formal analysis and structured treatment due to their tree-structure. Standard attack trees represent sub-goals that must be completed in a specific sequence, they have a hierarchical structure: the root

node represents the attacker's goal, which is further refined by defining sub-goals. As mentioned above, the sub-goals can be represented as sub-trees in the overall attack tree, where sub-trees, *i.e.*, sub-goals, are combined conjunctively or disjunctively.

While attack trees for purely technical attacks may be constructed by automated means [VNN14], for example by scanning networks and identifying software versions, this is currently not possible for attacks exploiting the human factors. Actually, only few, if any, approaches to systematic risk assessment take such "human factor"-based attacks into consideration. In this work we suggest the use of system models to systematically generate attack trees for attacks that may include elements of human behaviour. These attack trees can then be used as input to a traditional risk assessment process and thereby extend and support the brainstorming results. We extend previous work [KW13; KW14] by describing a systematic approach for the generation of attack trees from a system model. The generated attack trees are complete with respect to the model, that is, our method identifies all attacks that are possible in the model. This is achieved by basing the attack tree generation on invalidation of policies; policies in our model describe both access control to locations and data, as well as system-wide policies such as admissible actions and actor behaviour.

1.3 List of Publications

The work in this thesis has contributed to deliverables in the TRESPASS project and has also resulted in the following publications:

- Marieta G. Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller: "Externalizing behaviour for analysing system models" at *Managing Insider Security Threats (MIST)* 2013 [Iva+13]
  This paper introduces the externalisation of the behaviour in system models as a separate component, thus providing flexibility to the analysis to simulate different kinds of attackers.

- Jaap Boender, Marieta G. Ivanova, Florian Kammüller, and Giuseppe Primiero: "Modeling human behaviour with higher order logic: insider threats" at *Socio-Technical Aspects in Security and Trust (STAST)* 2014 [Boe+14]
  The paper aims at applying a fundamental theory from sociology in an attempt to model human behaviour. As a case study we present the modelling and analysis of insider threats in the context of an organisation.

- Michael Nidd, Marieta G. Ivanova, Christian W. Probst, and Axel Tanner: "Tool-based risk assessment of cloud infrastructures as socio-technical systems" at *The Cloud Security Ecosystem, Chapter 22* 2015 [Nid+15]
  This book chapter illustrates how we apply our modelling approach to a cloud environment seen as a socio-technical system.

- Zaruhi Aslanyan, Marieta G. Ivanova, Flemming Nielson, and Christian W Probst: "Modelling and Analysing Socio-Technical Systems" at *Socio-Technical Perspective in IS development (STPIS)* 2015 [Asl+15]
  The poster presents an overview of the modelling process, the attack generation, and a technique for further quantitative analysis of an attack tree.

- Marieta G. Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller: "Attack tree generation by policy invalidation" at *Conference on Information Security Theory and Practice (WISTP)* 2015 [Iva+15a]
  The paper describes the analytical approach to attack generation from a system model.

- Marieta G. Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller: "Transforming Graphical System Models to Graphical Attack Models" at *Graphical Models for Security (GraMSec)* 2015 [Iva+15b]
  The paper illustrates the graphical transformations applied on a system model in order to derive different attack vectors from that model, which are combined into an attack tree.

- Jan-Willem Bullee, Marieta G. Ivanova, Lorena Montoya, Christian W. Probst: "Literature Review on Socio-technical Security Models", In preparation for submission. [Bul+]
  A systematic literature review summarising work in the field of socio-technical security models.

1.4 Synopsis

For a better overview of this dissertation, a brief account of the chapters is presented below.

Chapter 2 starts with introducing the main background concepts. The core part of the chapter reviews existing literature on system modelling approaches, in particular those that focus on the attacker model and organisational infrastructure. In addition, the chapter reviews literature on policy languages, process modelling and attack generation techniques.
Chapter 3 briefly presents a running example, which we refer to throughout

the subsequent chapters. The chapter describes the concepts of our framework, focusing on externalising the human behaviour and describing the kinds of policies and the policy language used. Later in the chapter we illustrate them on the running example and conclude with a short discussion on the content of the chapter.
Chapter 4 presents the formalism behind the concepts described in Chapter 3. Using a variation of the Klaim language, we present the syntax of the process calculus. We also describe the semantics and the reference monitors used.
Chapter 5 describes the techniques of identifying attacks from the model. First, we illustrate graphical transformations of a system model to an attack tree. After that, we present the technique of generating the attack trees by invalidating policies in the system model.
Chapter 6 presents the cloud case study and how the techniques presented in this work are applied on it. Moreover, the work presented in this dissertation is compared to the literature reviewed in Chapter 2.
Chapter 7 presents some concluding remarks and depicts open questions and improvements as future work.

1.5 TREsPASS Project Acknowledgements

Part of the research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TRE$_S$PASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein.

Chapter 2

Related Work

In this chapter we describe related work on relevant topics. Before presenting it, we start with describing the background concepts.

In Section 2.2 we summarise the insights from a systematic literature review on socio-technical security models. It not only gave us the overview, but also served as a source of motivation for the research presented in this dissertation. Referring to the available work, that has been done in the area of organisational security, helped us identify the need of addressing the issues of the interplay between the technical and the social aspects of organisational vulnerabilities.

As policies play a vital role in our approach, both in the modelling part, and in the attack generation, in Section 2.4 we shortly present the types of access control policies and point the most popular policy specification languages.

Finally, in Section 2.5 we present existing work on attack tree generation techniques - a research area, that has been of increasing interest in the recent years both in academia and industry.

2.1 Background Concepts

In this section we introduce the core concepts used in this dissertation. We give simple examples where applicable and elaborate shortly on how these concepts are used in our work.

2.1.1 Socio-technical Systems

The term "socio-technical system" itself dates back to the end of World War II [ET60]. It has been realised that in an organisation it is not only about the technical part, but the social aspects are also important. Back then the concept was introduced in order to design an organisation in a way so it can unfold its potential and reach higher efficiency. Nowadays, seeing organisations as socio-technical systems is also beneficial for modelling the security of organisations.

Organisations constantly change and evolve. Some have hierarchical structures, while others have decentralised structures where each part is a semi-autonomous unit. However, no matter the structure, organisations always involve both the technical aspect as well as the social one.

Socio-technical systems pertain to theory regarding the social aspects of people and technical aspects of organisational structure and processes. "Technical" is a term used to refer to structure and a broader sense of technicalities. Socio-technical refers to the interrelatedness of social and technical aspects of an organisation.

Organisational security is a difficult problem in general and even more complicated when we talk about organisations as socio-technical systems. Many attacks, however, exploit organisational vulnerabilities on different levels. One of the many examples is the German steel mill incident, where a malicious actor has managed to infiltrate a steel facility [LAC14]. The attack involved other stages and required ICS knowledge, but the most crucial step turned out to be gaining access to the corporate network by using a spear phishing email, and respectively from there moved to the plant network.

Such examples clearly show that the human vulnerability possibly leads to a greater damage, which could sometimes even be catastrophic. There are many more examples reflecting the difficulty of securing socio-technical systems due to the interplay between the human factor and the technical aspect.

2.1.2 System Models

The term *system model* has a rather broad meaning. In the context of this dissertation we will use it for referring to an abstract representation of socio-technical systems formalised by an underlying process calculus. A given organisation can be presented by different system models, depending on the perspective. In this thesis we consider organisations as socio-technical systems, which include components as described below. In other words a system model describes all the relevant components of such a socio-technical system. These involve the physical infrastructure, the human factor, the policies, the processes, the items, and the data assets.

For a better overview, one could think of the system model's components addressing the following main layers:

- *physical layer* which represents the physical infrastructure of the organisation, in terms of buildings, rooms, doors, etc., and their interconnections. It is conceptually similar to a blueprint of the buildings of an organisation. This layer also includes physical items.

- *technical layer* refers to elements from the network domain (*e.g.,* computers, servers) as well as their logical connections. It also involves the processes, which represent the dynamics of this layer. Data assets belong to the technical layer too (*e.g.,* a file stored on a server).

- *access control layer* defines the access rules in both the physical and the technical layer. We model this layer with the use of policies. The corresponding policy specification language we use is described in Section 3.3.

- *social layer* involving the actors, together with their role in the organisations, access rights they have, their knowledge in terms of data, and items they possess.

With the above being said, we would like to note that throughout this thesis *a system model of an organisation* would then refer to a single instance of the system model, *i.e.*, a specific organisation modelled using our modelling approach. As a domain language for both the physical, the technical and the social layer, we use the process calculus presented in Chapter 4. We provide more elaborate descriptions of the system model components in the dedicated Chapter 3.

2.1.3 Security Policies

The American Heritage Dictionary from 1982 defines a *policy* as a plan or course of action designed to influence and determine decisions, actions, and other matters [82]. Consequently, in 2002, Bishop defines the goal of a *security policy* as to maintain the security of critical information, in terms of integrity, confidentiality, and availability of those information resources [Bis02].

Integrity ensures information is consistent, accurate, and trustworthy. *Confidentiality* is concerned with preventing sensitive information from being reached by the wrong users while making sure authorised users can get it. Finally, *availability* deals with guaranteeing reliable access to information by authorised people.

The above security properties are achieved by applying different security mechanisms, most often authentication, access control and auditing. *Authentication* is the process of ensuring that the user (subject) is the one it is declared to be. There are two types of *access control* - physical and logical. Regardless of the type, the technique is used to restrict access only to authorised users to view and/or operate the information. Finally, *auditing* of logs and records made primarily by the implemented security mechanisms facilitates after-the-fact analysis of security breaches and may be used to establish which entities are responsible for a breach.

In order to give a better overview of the different types of security policies, we refer to a framework proposed by Sterne [Ste91]. According to it, a security policy falls into one of the three categories mentioned below:

- *security policy objective* which may be considered an overarching goal or a "mission statement" for information security. A security policy objective defines, at a high level, which information resources to protect and what to protect them against; it does not prescribe specific protection mechanisms or describe technical details of attacks. An example of a security policy objective might be 'maximum network availability should be maintained at all times'. While these statements are important, they are very high level and provide limited opportunities for further analysis in the context of this dissertation.

- *organisational security policy*, which delves into more details with security rules, mechanisms, and practices in order to support the security policy objectives set out by an organisation. The organisational security policies are also a natural place to import, implement, and codify all the relevant legislation and compliance measures related to managing and protecting an organisation's information resources. This requires defining criteria

for authorising individual users, user roles, conditions for delegation of authority, etc. An organisational security policy is meaningful as long as it provides individuals reasonable ability to determine whether their actions violate or comply with the policy. Continuing the example above, an organisation might declare that only senior staff members should be given a key to the company premises. This would be supporting the security policy objective of maintaining maximum network availability at all times because it limits the number of people potentially having physical access to the building in which such network equipment is located.

- *automated security policy*, which operates at the (low) technical level and involves technical measures and mechanisms employed in order to implement and support organisational security policies. Examples include the access rules defined in a network gateway or firewall to enforce network separation and control.

Policies play a central role in the modelling approach as well as the analysis and attack tree generation presented in this thesis. In Section 3.3 we describe how we relate the aforementioned categories of security policies in the policy-specification language we present.

2.1.4 Attack Trees

Attack trees represent attacks in a hierarchical structure with the goal of the attack being the root of the attack tree. Leaves represent the basic actions of attacks, while the intermediate nodes are sub-goals combining the basic actions either conjunctively or disjunctively. The "OR" nodes describe different alternatives, *i.e.*, one satisfied sub-goal is enough, and the "AND" nodes describe the steps needed for a successful attack, *i.e.*, all the sub-goals of an "AND" node should be satisfied for the goal to be achieved. In this dissertation every node is an "OR" node unless it is marked as an "AND" node with the help of a bent line.

Attack trees are widely used both in industry and academia due to their broad usability. They are informative and descriptive, accessible for non-experts, while at the same time they can be assigned a formal semantics that allows scientific analysis. Based on the analysis, practitioners can then define actions which can reduce or eliminate vulnerabilities.

In Figure 2.1 we show a simplistic attack tree. The root defines the goal of the attack, namely, to steal a treasure from a bank. The goal is achieved by

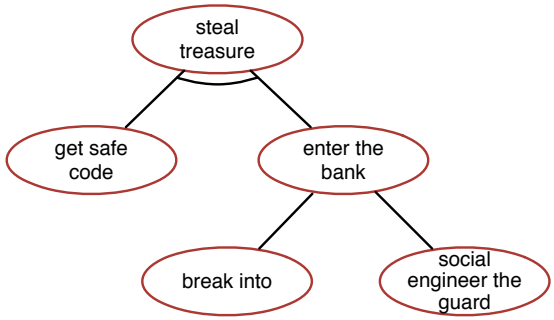


Figure 2.1: How to steal a treasure

getting the combination of the safe and either breaking into the bank or social engineering the guard.

2.2 Socio-technical Security Models

The content of this section is based on a structured literature review in the area of security modelling [Bul+]. In the review we address the research question: *"What features do current security models have?"*. We shortly elaborate on the methodology of the literature review and present a summary of the results.

2.2.1 Methodology

We have performed a systematic literature review [Kit04] to identify relevant work on socio-technical security modelling languages, and to summarise the current state of this topic and highlight the challenges. A review protocol describing each step of the review, including eligibility criteria, was developed before beginning the search for literature and the data extraction.

We considered articles covering aspects of security modelling of socio-technical systems and keywords from the results of an initial search [Pie11a; SEH12; DC04; DC05; Dra06; HP11; PH08; PH09b; Sam11; DFP98a; GP03; BLP02; PHN07a; Mat+05; Mat+08; FLE09; DPH10b; Dim12; SBM03; Sco04]. These articles covered 9 models, containing together 134 keywords, from which the most relevant were selected and combined, to increase relevance of the query results. The decisions on articles, keywords, and their combination were made after intensive discussions with experts inside and outside of TRE$_S$PASS. As a

result ten keywords were identified. The two most important keywords turned out to be *socio-technical* and *security modelling* and the rest being as follows: *attack, cyber, cyber-attack, insider, model, scenario, vulnerability* and *vulnerability analysis*. In the end, we formed 28 search queries consisting of the two most important keywords as well as a combination of two other keywords.

The inclusion criteria were defined in two parts. The first one requires the study to be written in English, as it is the language being favoured by the Scientific Community when it comes to published research work. The second inclusion criterion, being established as directly answering the research question, requires the study to deal with aspects of security modelling of socio-technical systems. We identified a socio-technical security model as having the following characteristics:

- models a part of an organisation
- takes at least two of the following into account: social, technical, business or physical/spatial aspect, and
- has an attacker component, *e.g.*, goal of an attack, probability of breaking a component or distinguishing between benign and malicious users.

The search was applied to the SCOPUS database, which also covers publications from Cambridge University Press, Elsevier, Springer, Wiley-Blackwell and the IEEE. The automated search was carried out in 2013 and 2014, and the results of the queries were filtered based on a first read of the article and an assessment of inclusion criteria. We also scanned the reference lists of the papers in order to identify relevant other sources. In addition we have interviewed domain experts within TRESPASS and outside of the project, who have suggested examining some additional studies. Even after querying the database, the domain experts kept sending us studies to be potentially included in the analysis.

The rest of the sections in this chapter present a summary of the results obtained. An overview of the results can be found in Table 2.1.

2.2.2 Representation

The approaches considered in this review represent models either graphically or textually. The graphical models can be divided in tree and graph structures, diagrams, and map overviews.

The tree structured models include the work by Dragovic et al. [DC05] and Scott *et al.* [Sco04], where trees represent the world, and the MsAMS framework [FLE09], where trees represent network topologies. Attack trees model all steps that need to be taken to achieve the main goal of the attack [KOS13; PDP13a; TML10; VVM12]. Alternatively, attack patterns describe generic approaches used by attackers [KOS13]. Finally, fault trees are used to represent failure information about systems [VVM12]. Boolean logic Driven Markov Processes are an extension of Fault Tree with Markov processes [KBP12].

Graph structures are used to construct Capability Acquisition Graphs, presented by a tuple consisting of Vertices, Edges and System properties [Mat+08]. In the CySeMol model, the graph structure is used in a Reachability graph to link steps in an attack [SEJ10; SEH12]. Hyper graphs are used in the ANKH model to represent membership of a group [Pie11b] and in the MsAMS language to define broadcast communications [FLE09]. Portunes [DPH10b; Dim12] and ExASym [PHN07a; PH08; PH09b] use directed graphs to connect data or places of interest in the model. A totally ordered graph is used to represent coordinated attacks [Sam+13]. Directed Acyclic Graphs (DAG) are used to model Attack Graphs, containing paths an attacker could use to achieve his goal [Xie+09; Sar+14], and directed bipartite graphs are used to visualise Petri Nets, and are well suited for modelling distributed systems and concurrent behaviour [SF93].

Diagrams also come in different flavours. Attack sequence diagrams describe an intrusion from the intruder's point of view as a sequence of ordered steps. Each step describes an attacking activity to be used [KOS13]. Data flow diagrams are used to create threat models [Sho08]. Misuse case maps focus on vulnerabilities, threats and intrusions from an architectural point of view, and are an extension of Use Cases, with security elements [KOS13]. The related Misuse sequence diagram graphically shows an intrusion sequence as a combination of misuse cases and UML sequence diagrams, helping to analyse complex intrusion scenarios [KOS13].

Finally, ExASyM uses a building blue print as basis for the model [PHN07a; PH08; PH09b; Sam11]. Misuse case maps present security issues from an architectural perspective. They combine perspectives from misuse cases and use case maps, providing a combined overview of a software system's architecture and its behaviour by drawing usage scenarios paths (aka use cases) [KOS13].

Textual notations for models are often used for elements in models, for example processes. The algebraic process calculus is an overview description of the processes and communication of a system. This allows formal reasoning about behaviour and the system. One example of an algebraic process calculus is KLAIM, the Kernel Language for Agents Interaction and Mobility [DFP98a], which is the basis for both ExASyM [PHN07a; PH08; PH09b; Sam11] and Por-

tunes [DPH10b; Dim12]. The situational calculus is used to model and analyse coordinated attacks [Sam+13], and temporal logic allows to reason about time aspects [Sha+10].

2.2.3 Infrastructure

The vast majority of the studies that represent infrastructure (16 out of 20) cover the digital layer in their modelling approach; a big portion of them model both the physical and the digital world.

Ten *et al.* [TML10] deal with cyber-security of critical network infrastructures. The study proposes a supervisory control and data acquisition security framework with four major components: real-time monitoring, anomaly detection, impact analysis and mitigation strategies (RAIM). Xie *et al.* [Xie+09] focus on analysing network security vulnerabilities, therefore they consider the digital layer of an infrastructure. Shahriari et al. [Sha+10] apply an actor-based language using reactive objects (REBECA). The study deals with network security on the Transport Protocol Layer. It models a typical network including client and server. Aiming to diagrammatically represent complex hacker attacks from multiple perspectives, the Hacker Attack Representation Method (HARM) by Karpati *et al.* [KOS13] uses a combination of 6 modelling techniques. With the help of the Misuse Case Maps (MUCM) the system architecture targeted by a specific attack is modelled. Dragovic *et al.* [DC04; DC05; Dra06] work in the field of information security and privacy protection in ubiquitous computing. They model the world unifying the physical and the virtual realms. Each instance in this world belongs to a container class, which can be physical, intermediate, or virtual. The notion of infrastructure in ExASyM [PHN07a; PH08; PH09b; Sam11] is represented as set of locations and connections. The physical layer describes the architectural plan of the organisation being modelled, *e.g.*, how rooms are connected with each other. Similar to the physical layer, ExASyM models network components and the connections between them. The MsAMS modelling framework [FLE09] focuses on modelling networks and is based on ambients that represent hosts, services, vulnerabilities, networks, users, and even credentials. The social layer is also described as ambients interacting with each other. Mathew *et al.* [Mat+05; Mat+08] model information about the physical location and reachability of information assets on a network. Even though the study is focused on network security, it considers both the network infrastructure and physical aspects. Samarji *et al.* [Sam+13] focus only on modelling system networks. Sommestad *et al.* [SEJ10; SEH12] model information systems.

Only four of the modelling approaches in this literature review reflect all three

types of infrastructure in their studies. Pieters et al. [Pie11b] model the physical and digital infrastructures and also reason about access in system models including human actions, and their graph-based reference model reflects all physical, digital and social infrastructures. Scott *et al.* [Sco04] model the world as a nested tree of entities, similar to ambients in the Ambient Calculus. Sorts are used as constraints for how entities could be nested. The authors take into account the physical world, the digital world, though modelled as physical objects, as well as the actors, modelled as autonomous physical entities. In Portunes [DPH10b; Dim12] the world is also divided in the physical, digital, and social layer. A later study by Pieters *et al.* [PDP13a] is focused on alignment of policies from different domains: access control, network layout, and physical infrastructure as well as the social domain. While the study is not focused explicitly on modelling these domains, they are still part of it as components of the policies being aligned.

The only socio-technical model that focuses exclusively on the physical domain is the STS model by Lenzini et al. [LMO15]. This approach models the infrastructure as a graph structure that gives rise to a labelled transition system (LTS) capturing the infrastructure state, and evaluates security properties directly on this LTS.

2.2.4 Assets and Containment

ExASyM [PHN07a; PH08; PH09b] considers the objects that the actors work with or any data in general, be it located at actors or accessible at certain locations. Pieters *et al.* [PDP13a] consider the assets of an organisation described by high-level policies ("sales data should not leave the organisation") as well as desirable and undesirable states of those assets ("being in the hands of competitors"). In low level policies individual actions of actors are constrained ("this door can only be opened with a specific key"). An earlier study by Pieters *et al.* [Pie11b] faces an issue with the containment approach in the case when there are different domains represented (physical, digital, and social) and the physical and digital assets being modelled are combined. Assets in Portunes [DPH10b; Dim12] can belong to the physical or digital domain, *e.g.*, a usb dongle and service data. Ten et al. [TML10] address cyber-security of critical infrastructures, especially electrical power infrastructure, thus they consider cyber-assets of the power infrastructure including computer and communication devices installed in power plants, substations, energy control centres, etc. Xie *et al.* [Xie+09] model network resources as assets. In the study by Dragovic et al. [DC04; DC05; Dra06] the modelling of assets exhausts with modelling data objects. Ambients in the MsAMS modelling framework [FLE09] are the key components when modelling the world thus they are abstractions, which,

among others, also represent assets. The studies by Mathew *et al.* [Mat+05; Mat+08] are focused on information assets in a network. They refer mostly to critical files, which are called "jewels". When modelling coordinated attacks, the study by Samarji *et al.* [Sam+13] allows resource sharing between attackers, therefore different assets of a system could be threatened at the same time. Sarkar *et al.* [Sar+14] model assets in the form of data or artefacts, and annotations. In [SEJ10; SEH12] assets and their relation to each other are specified and risk is estimated with regards to the assets in terms of probabilities (architectural meta-model and probabilistic dependencies).

We consider containment either as the containment of an object at a location, or an object within another object. An example of the latter is a hard disk within a PC.

Objects and actors can be modelled to be at a location. In this case, actors also can travel within the infrastructure, gaining objects or performing actions [Pie11a; PHN07a; PH08; PH09b; DPH10b; Dim12; Sco04]. Similar to the real world, actors can only travel within the physical infrastructure. An actor can for example go to a room and get some object [PH09b], but not the bits of digital file. However, there can be interaction with objects in the digital infrastructure (*e.g.,* by using a computer to start a process).

The second meaning of containment is an object within another object, whereas the relationship between objects is more in the hierarchical sense. Such a kind of relationship can be modelled by some of the approaches considered [KOS13; DC04; DC05; FLE09; DPH10b; Dim12; Sco04; LMO15]. Examples include a room within a building or a PC within a room, as well as the containment of a digital object in a physical object [DC04; DC05; DPH10b; Dim12; Sco04]. Clearly this containment of digital objects in physical objects cannot be reversed, that is data objects can not contain physical objects. Another approach is to model everything as an ambient [FLE09] and use nesting. A company network would be an ambient, containing other ambients, such as PCs, firewalls and network routers.

2.2.5 Processes, Actions and Behaviour

Processes are generally defined as a sequence or flow of steps or actions. In the context of socio-technical modelling this is a sequence of attack steps [KOS13; Xie+09; Sam+13; PDP13a; SEJ10; SEH12; Zha+11; FLE09] or the (data) flow through a system or application [KOS13; Sho08; Sar+14].

Processes are represented as part of a model in ExASyM [PHN07a; PH08;

PH09b; Sam11], Portunes [DPH10b; Dim12] and Scott *et al.* [Sco04]. In the model of Scott, software model checkers (*e.g.,* Promela) ensure that processes are being free of deadlocks, race conditions and that liveness properties hold.

By using attack trees, processes are used in a different way. The path through the tree is a sequence of attack steps and therefore an attack path can be seen as a process [TML10]. Extending the tree with Markov Processes ensures that succeeding attack steps are executed [KBP12].

Actions Activities related to computing can be put in the environment of an Ambient, including hosts, services, vulnerabilities, networks, users and credentials [FLE09]. The activities of mobile agents that react to changes in the context are described by [Sco04], actions involving mobile agents are expressed in: Out, In, Read, Eval and NewLoc [DFP98a].

Regarding vulnerabilities to the system, these are described in [TML10; Xie+09; Zha+11]. Specific intrusion sequences, including interactions and message sequences are used in [KOS13].

Behaviour The expression of human behaviour in general is described in terms of actions [Sco04; Pie11b] and how the user interacts with the system and what processes are involved [KOS13].

Meta-attacks are described as attacker behaviour on a system, *e.g.*, database searches or unusual file deletion [Mat+08] or the expected behaviour and actions an attacker must perform to achieve the goal of the attack [VPH12; KBP12; KOS13; VVM12; SEJ10; SEH12; Zha+11]. Also specific behaviour is described, *e.g.*, actors moving between locations in a physical infrastructure [DPH10b; Dim12; PH08; LMO15]. The actors in the model can perform actions (*e.g.,* change location or store data) [PH08], or move assets [DPH10b; Dim12; LMO15]. Furthermore, attackers can start processes [Sha+10] and it is assumed that they will pick attack steps that are related to their skills [Xie+09].

2.2.6 Actors

In ExASyM [PHN07a; PH08; PH09b; Sam11] actors can move in the infrastructure by following the connections between the locations. In ANKH [Pie11b], on the other hand, humans and non-humans are treated symmetrically. There is no need to distinguish between actors, objects, and credentials a priori. In the

MsAMS framework [FLE09] basically everything is represented through an ambient, including the users. Scott *et al.* [Sco04] model actors as autonomous physical entities with the ability to move between rooms. Mathew *et al.* [Mat+05; Mat+08] model users with different roles in order to evaluate their influence on a network and detect possible violations. Since Karpati *et al.* [KOS13] represent details about the actors in the system architecture in Misuse Case (MUC) diagrams, a colour notation is used to distinguish between "normal actors" (or "regular users") and the attacker. Actors in Portunes [DPH10b; Dim12] are allowed to move objects around and thus modify the graph representing the system. Actors are also able to interact with each other. Samarji *et al.* [Sam+13] present the system in terms of predicates. The subject of an actor's predicate is always the ID, uniquely identifying actors. Actors in DASAI [Sar+14] can be humans or automated agents. The agents in the system are assumed to be insiders. There is also the possibility to model interaction between colluding agents. In contrast, actors in [SEJ10; SEH12] are modelled as part of the architecture, regardless of whether it is an outsider or insider. In their work Dragovic *et al.* [DC04; DC05; Dra06] deal with information exposure threats where the threat does not include a malicious intruder. In the STS model actors, including the intruder, can act probabilistically and perform different actions (*e.g.,* move or lock an object) [LMO15].

2.2.7 Policies

Low level policies manage accessibility within an infrastructure. In this sense they describe direct actions being allowed if certain conditions are satisfied.

ExASyM uses access control policies at locations. Actors need to comply with the credentials in order to be able to perform the allowed actions specified in the policy. The network attack model of Xie et al. [Xie+09] consists of attack states, attackers, and attack rules. The attack rules describe the transitions between attack states and define preconditions. For optimisation purposes, Dragovic et al. [DC04; DC05; Dra06] assign policies to a given container class. In this way a policy applies for each instance of the class thus avoiding unnecessary repetition. Their studies deal mostly with policies concerning access control and authorisation. As the world in the MsAMS framework [FLE09] is based on ambients, the policies are embedded in the rules of the ambient. Samarji et al. [Sam+13] do not define explicit policies. However, there is an implicit approach by defining predicates, modelling the assets and the knowledge of actors.

High level policies describe actions at an abstract meta-level. An example of a high level policy is "*all behaviours that have an undesirable outcome*".

Pieters *et al.* [PDP13a] focus on formally identifying misalignments between the different levels of policies, for example, access control policies and organisational ones. Scott *et al.* [Sco04] consider mobility policies as well as global security policies. A potential problem of conflicting policies is encountered and a solution is proposed by describing suitable conflict resolution meta-policies. In contrast to the majority of studies in this literature review, where policies are used in order to ensure security and often attacks are derived by enforcing the policies, in this study policies are used for controlling Sensient Mobile Applications at runtime as well as making the development of such applications easier. The Portunes modelling language [DPH10b; Dim12] expresses policies from physical and digital security by low level policies, and then introduces high level policies in terms of security awareness.

2.2.8 Quantitative Measures

Quantitative measures are used to annotate model elements either during model building or as a result of computations. The models can be annotated with properties related to attackers and properties related to the owners of the system. An important measure considered in studies is the probability of success of a launched attack (step) [VVM12; SEJ10; SEH12]. In terms of risk management, the impact of an exploited vulnerability [Zha+11] and organisational impact [Xie+09] of attacks are of interest.

Properties considered related to attackers include annotations of monetary costs needed to perform an attack [KOS13; VVM12; Xie+09], the time needed to execute an attack [KBP12], the needed skill for an attack [KOS13], vulnerability exploitability of a system [Zha+11], and the necessity for special tools [VVM12]. Perhaps the most valuable annotation for an attacker is the risk of detection [KOS13; Xie+09].

2.2.9 Attacks, Vulnerabilities, and Countermeasures

ExASyM recognises possible attackers based on the analysis of the model and presents them as sequence of actions.

Pieters *et al.* [PDP13a] provides the basis for existing and future methods for finding security threats induced by misalignment of policies in socio-technical systems. Attacks are generated from mismatches between global policies and local ones. An attack is considered again as a sequence of actions.

Ten *et al.* [TML10] evaluate system-, scenario-, and leaf-level vulnerabilities by identifying the system adversary objectives. In their anomaly detection they use event correlation techniques that are categorised as temporal, spatial, or hybrid. The impact analysis evaluates the consequences of cyber-attacks on SCADA. Mitigation strategies introduce security improvements of the most vulnerable components of an attack scenario (presented as sequence of events).

Xie *et al.* [Xie+09] present an automatic generation of attack graphs. The attack graph framework includes a host access graph and sub-attack graphs. Each individual sub-attack graph presents the attack scenarios from one specific source host to another specific target host. The host access graph presents the access relationships between each pair of hosts.

Mathew *et al.* [Mat+05; Mat+08] use a static analysis tool to periodically construct Capability Acquisition Graphs (CAGs) which are then analysed to uncover any possible attacks. Information about vulnerabilities in network services is provided beforehand as an input to the tool. As the CAGs are generated periodically, there is potential for mitigation of attacks in the form of raising an alert when an unauthorised privilege accumulation becomes apparent.

Shahriari *et al.* [Sha+10] show how an attacker can combine simple attacks into multiphase attacks. The study uses a model checker for finding counterexamples as violations.

The $ST(CS)^2$ platform [AK12] aims to provide its users with guided cyber security warnings based on the subscriber's socio-technical security posture. As opposed to the general cyber security warnings, which give only an overview of the current situation, the authors talk about guided security warnings where the threat level and the recommended countermeasures are customised depending on the user's socio-technical posture.

In another study vulnerability is modelled in the form of possible step-wise attacks [Pie11b]. An attack is successful if the attacker gets access to a designated asset.

Dragovic *et al.* [DC04; DC05; Dra06] focus on a subset of information leakage threats, also called information exposure threats. In their system for autonomic context-adaptive security, they focus on reasoning about the context. The reduction of the Level of Exposure (LoE) for all data objects is achieved by two main protective actions: containment manipulation and information reduction.

The vulnerabilities are provided as an input component in MsAMS modelling framework [FLE09]. Once the network is modelled, an attacker, who is also represented as an ambient as all other components, is simulated dynamically.

In this way an attack path is found, which is allowed by the modelled ambients and their embedded rules.

Different approaches of modelling complex attacks from different perspectives are used in HARM modelling technique [KOS13]. The study provides an integrated view of security attacks and system architecture - misuse case maps and misuse sequence diagrams.

In Portunes [DPH10b; Dim12] attacks are generated by finding inconsistencies between the security policies in the different domains (physical, digital and social). Respectively, an attack scenario could combine physical, digital and social means of achieving his/her goals.

Samarji *et al.* [Sam+13] derive individual, coordinated (simultaneous) and concurrent attacks from the model. There are also types of attackers' collaboration: load accumulation, load distribution, role distribution. The study formally describes attacks by presenting the system state in terms of predicates. The authors have chosen a pessimistic approach: in coordinated attacks, if a given knowledge is required, it is enough that one of the actors has this knowledge.

In the study by Sommestad *et al.* [SEJ10; SEH12] vulnerabilities are threats defined by domain experts as part of the model (both the abstract and the concrete). An abstract model is defined as a base for a concrete model. Additionally a meta-model is associated with a probabilistic model for evaluating the security risk. Countermeasures are modelled with the aim of minimising the risk. The study is focused on monetary loss from assets, but other application domains are also possible.

In Sarkar *et al.* [Sar+14] the vulnerabilities are defined by domain experts and serve as an input to the analysis tool. An attack model is first made by a domain expert. Attack A is successful on a process P when there is a mapping relation from A to P with certain conditions being satisfied, *i.e.*, an attack is successful if there is a "similarity match" between A and P. Countermeasures work as follows: once an attack is found, improvement points in the process are automatically identified (sorted by how heavily a certain step is attacked). P is then evaluated to check whether the improvement was successful.

The STS model allows to evaluate security properties, such as the minimal cost or the maximal probability of the intruder reaching a sensitive location or object, using the probabilistic model checker PRISM [LMO15].

Study	Representation	*Physical* Infrastructure	*Digital* Infrastructure	*Social* Infrastructure	Assets	Containment	Processes	Actions	Behaviour	Actors	*Low Level* Policies	*High Level* Policies	Quantitative Measures	Attacks	Vulnerabilities	Countermeasures
[TML10]	ADtree	-	+	-	+	-	+	+	-	-	-	-	+	+	+	+
[Xie+09]	Graph	-	+	-	+	-	+	+	-	-	+	-	+	+	+	+
[Sha+10]	Text	-	+	-	+	-	-	+	-	-	-	-	+	+	-	-
[KOS13]	Multi	-	+	-	+	-	+	+	-	+	-	-	+	+	+	-
[DC04; DC05; Dra06]	Tree	+	+	-	+	+	-	-	-	-	-	-	+	+	-	+
[PHN07a; PH08]	Graphical	+	+	+	+	-	+	+	+	+	+	-	+	+	+	+
[FLE09]	Multi	+	+	+	+	-	-	+	+	+	+	-	+	+	+	-
[Mat+05; Mat+08]	DAG	+	+	-	+	+	-	+	+	+	+	-	+	+	+	-
[Sam+13]	Graph	-	-	-	+	-	±	-	-	+	±	-	+	+	-	-
[SEJ10; SEH12]	UML like	-	+	-	+	-	+	+	-	-	-	-	+	+	-	-
[Sco04]	Tree	+	+	+	-	+	+	+	-	-	+	+	+	+	+	+
[DPH10b; Dim12]	DAG	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
[PDP13a]	Text + Venn	+	+	+	+	-	+	-	+	-	+	+	-	+	-	-
[Sar+14]	DAG	-	+	+	+	-	+	-	+	+	+	-	-	+	+	-
[AK12]	UML like	-	-	+	+	-	±	-	-	+	-	-	-	+	+	+
[Pie11a]	HyperGraph	+	+	+	+	+	-	+	+	+	+	+	-	+	-	-
[Zha+11]	Graph	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[KBP12]	Graph	-	-	-	-	-	±	-	-	-	-	-	-	-	-	-
[DFP98a]	Text	+	+	-	+	+	+	+	+	+	+	+	+	+	+	+
[VVM12]	Tree	+	±	-	-	-	-	-	-	-	-	-	-	-	-	-
[SYE10]	Graphical	-	±	-	-	-	-	-	-	-	-	-	-	-	-	-
[Mai+04]	UML like	-	-	-	-	-	±	-	-	+	±	±	-	-	-	-
[Fra06]	Graph	-	-	-	+	-	-	+	+	-	-	-	+	+	-	-
[MKM09; Moh10]	Graph	-	-	+	+	-	+	+	+	+	+	+	+	+	±	±
[LMO15]	Graph (LTS)	+	-	+	+	-	-	+	+	+	-	-	+	+	+	-

Table 2.1: Models and their characteristics, one model can contain multiple references

2.3 Kinds of Attackers

Bishop *et al.* emphasise that the distinction between *insider* and *outsider* is not a result from a binary function [Bis+10]. Instead, it is more realistic to distinguish between different kinds of attackers with respect to the level of insiderness. These levels are based on different parameters such as access, knowledge, and trust [Bis+10]. Mundie *et al.* also introduce different components in their attempt to define an ontology for insider threat [HMP13].

The system models discussed in Section 2.2 consider only one single kind of attackers. It is assumed that they know everything, *i.e.*, they represent the strongest possible attacker with respect to the model. This assumption is partly justified by the fact that it is very hard to collect data to explain human behaviour. However, as discussed above, assuming a strongest possible attacker also means that the analyses on system models will deem most organisations to be vulnerable to most attacks. This happens because an attacker with legal access to large parts of the organisation, such as a CEO or a cleaning lady, also has the possibility to attack large parts of the organisation. In real life this problem is solved by, *e.g.*, trust or background checks. In the analyses of system models, we need to assume that the actor might perform the actions, thus raising an alert. Therefore we need other concepts in system models to solve this problem [PH09a].

The Dolev-Yao attacker is the most powerful attacker when talking about protocol analysis [Cer01]. In the formalisms described above, the insider knows how to get to any location, for example what key is needed to open a certain door, where the key is located, and how to get it. In this case, we can think metaphorically of a Dolev-Yao insider. However, such a kind of insider defines the upper bound of the attacker's abilities, which is not realistic in real life. In addition, modelling such an attacker would make the system vulnerable to most kinds of attack thus making the model useless. Instead we would like to define different types of actors, *i.e.*, actors with typical kinds of behaviour.

Some studies already exist that could provide the data to define typical kinds of attackers. Magklaras *et al.* classify insider misuse as either intentional or accidental [MF01]. A study shows that accidental security incidents by insiders happen more often than malicious insider attacks [Gra09]. Existing research on personality traits in relation to insider threats introduce a classification of the insiders based on motivational categories. For instance, one category in the topology is the explorer type. Driven by curiosity, they are benign and often perpetrate without realising an attack [SPR99].

By defining separate groups of attackers it would be then possible to examine

how vulnerable a system is towards different behaviours. One relevant evaluation parameter is, for example, the likelihood of social engineering [MS03].

2.4 Policy-specification Languages

In this section we provide a short insight on the state-of-the-art of policy specification languages. For a more detailed discussion of their different features and properties, we refer the reader to the relevant citation.

Most often in the literature access control policies are divided into three major groups: discretionary, mandatory, and role-based.

Discretionary Access Control (DAC) restricts or permits access to objects through an access control policy determined by the object's owner group/subject. A typical example of *DAC* is the UNIX file mode.

Mandatory Access Control (MAC) enforces access based on regulations by a central authority (*e.g.,* an operating system) and thus cannot be altered by an end user.

Role-based Access Control (RBAC) as self-explaining, regulates the access by using different roles, to which users belong to. An example could be any big enterprise with a stable organisational structure.

The role-based access control policies are further refined into the following subcategories: *trust-based access control (TrustBAC)* [CR06], *delegated role-based access control (DeRBAC)* [CK06; CK08], *risk-aware role-based access control (R2BAC)* [CC12], *risk-adaptive access control (RAdAC)* [KSB11], and *attribute-based access control (ABAC)* [Hu+14].

Among the most widely used policy-specification languages are XACML [XAC13], DPL [LBN99], MRPL [Sco04], SPDL-2 [SBM03], SWIL [Sco04], PEAL [CHM13], and Cassandra [BS04].

2.5 Attack Generation Techniques

As mentioned earlier, attack trees are widely used as a basis for automated risk assessment tools. However, currently they are manually constructed by the

domain experts relying on their knowledge and experience. While this could work for small attack trees, when applied on big organisations, it becomes a tedious and error-prone process. An automated attack tree generation can give the practitioners large and correctly constructed attack trees, which are also complete with regards to the underlying system model. Moreover, an automated generation of attack trees enables the opportunity of reiterations in case of system updates and changes.

There are some studies tackling this problem. In this section we summarise the research we came across with, which addresses different aspects of attack generation techniques, and more precisely generation of attack trees.

Attack representation models include attack graphs, attack trees, and variations of attack trees: attack-defence trees, fault trees, etc. A major flaw of the attack graphs is the state space explosion. A naive approach for an automatic generation of attack trees, on the other side, is exponential in number of nodes.

A study by Kotenko *et al.* considers both technical and social aspects in security. However, it is more focused on information security, *i.e.*, the technical part is software-related. Even though there is a notion of physical access in the constructed attack vectors, it is restricted only to control areas, neglecting the physical infrastructure of the organisation in question [KSD11]. A serious problem, which the study has, is the exponential complexity of the security analysis, which drastically decreases its usability.

Due to the lack of adequate attack tree generation techniques, combined with the benefits from such an automation, this research gap has drawn great attention and has become attractive for both researchers and industry practitioners [HKT13; VNN14; Pau14; PAV14; PAV15]

Hong *et al.* try to tackle the scalability problem using logic reduction techniques in order to simplify the attack tree representation [HKT13]. The study proposes two techniques. The first one is Full Path Calculation, where similar nodes are grouped together. The second one is Incremental Path Calculation, where the attack paths are recursively expanded in order to avoid node repetition. The simulations done successfully confirm the size reduction of the attack trees, though there is a trade-off between construction time and memory usage. Moreover, in a system where there are often updates, the overhead is repeated every time, which could be time and computationally expensive. The biggest issue, however, is the flattened structure of the attack trees, making the further exploitation hard, if not impossible.

In their work, Pinchinat *et al.* synthesise attack trees based on a high-level description of the system [PAV14]. They support high-level actions, playing

the role of sub-goals, being further refined in the successive nodes in the attack tree. Using GAL (Guarded Action Language), attack trees are then generated using model checking. Follow up work from the same group introduces their tool to support the earlier research [PAV15]. Even though high-level actions help to reduce the complexity, it seems the scalability problem is still present due to the combinatorics induced by the merging operator when constructing the attack trees.

Chapter 3

System Modelling

In this chapter we first introduce a running example on which to demonstrate the modelling approach (Section 3.1), followed by a description of the model components and their roles (Section 3.2).

As policies and processes play a key role in our system model, separate sections are dedicated to them (Section 3.3 and Section 3.4, respectively). Throughout the chapter we will refer to the running example in order to illustrate the model components.

Finally, we discuss the work we have done in the direction of modelling the human behaviour component in Section 3.5.

3.1 Running Example

Throughout this and the following chapters we will use a scenario from the IPTV case study to illustrate our concepts [D1.3.2]. This case study describes a home-payment system designed to support elderly people or people with disabilities, who may have difficulty in leaving their home, in managing their own money. Considering the target group, the system is integrated within an already familiar

device, which makes it easy to use, and that is the television set.

An overview of the architecture of the system modelled in this case study is shown in Figure 3.1. Due to confidentiality reasons, the technical details are omitted, but the main features of the original case study have been retained. The work presented in this dissertation has been applied on the original system where similar results were obtained.

The architecture for delivering the service via the television is based on using contact-less payments cards to authenticate users and provide payment capabilities via a television remote control. The TV remote control is paired with an IPTV set-top box, in order to support both the security and power management requirements of the solution. The design offers the opportunity for people who are not familiar or comfortable with mobile technology to receive the benefits of the ever-increasing range of mobile services in their homes via their television screen.

This case study provides the opportunity for exploring various security aspects, ranging from the pure technical concerns, such as information transmission and storage, to socio-technical ones, dealing with the use of and interaction with the technology. Although this system could offer great convenience, it also has the potential to expose the account holder to significant social risks.

The design behind the service has been adopted in order to support a range of use cases. In the rest of this thesis we will be using one of the scenarios of the IPTV case study. In particular we will explore its socio-technical features so as to illustrate and validate the methods and tools we have developed.

3.2 Components Overview

We start by introducing the main concepts of a language, most of which are illustrated on the running example in Section 3.6 below. The syntax and the semantics of the formal language are covered in the following Chapter 4. The model represents the infrastructure of organisations – both the physical and the digital world – as nodes in a graph. These nodes can represent rooms, access control points, and similar locations. Locations that are physically connected are linked by directed edges in the graph, while logically connected locations are connected with undirected edges. A location may belong to several *domains*, *e.g.*, a location can be part of the building and the network. In the model, domains are used to limit where processes can move, *e.g.*, human actors are restricted to (physical) room nodes, computer processes are restricted to

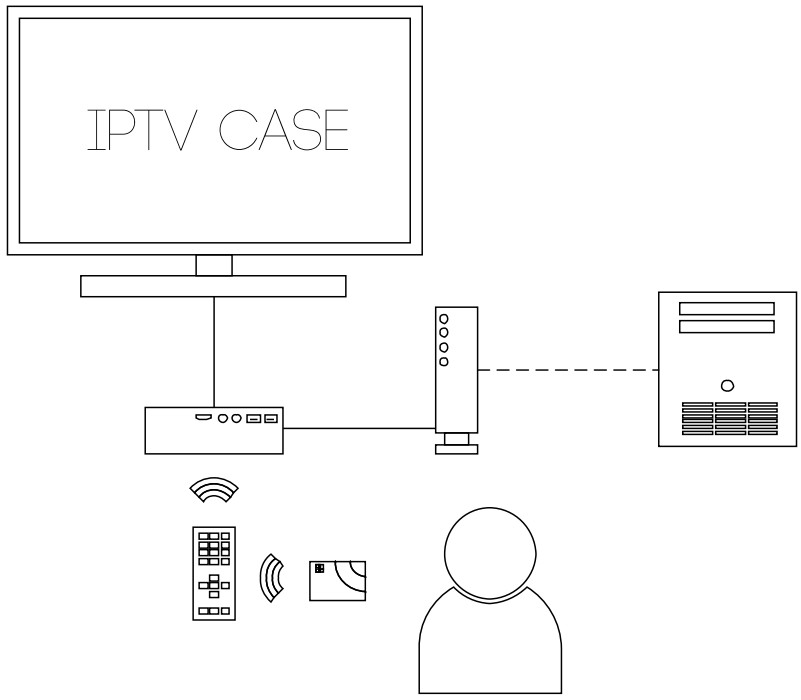


Figure 3.1: Physical system model of IPTV system

network nodes. Beyond these locations, the model has many other components:

Actors are also represented by nodes in the model and they move around in the infrastructure. This also allows actors to share roles that can be used in defining and checking policies.

Assets are used for modelling any kind of data or items, that are relevant in the modelled organisation. Assets are represented by nodes that can be attached to locations or to actors; assets attached to actors move around with that actor.

Actions are performed by actors on a target, which can be any location in the model, including physical locations or actors.

Policies are used in the model in a rather broad sense; they represent both regulation of access to locations and data, and the behaviour as expected by an organisation from its employees. Policies consist of required credentials and enabled actions. Credentials represent what an actor needs

to provide in order to enable the actions governed by a policy. Enabled actions represent the actions an actor is allowed to perform upon providing the required credentials. Each policy can consist of several pairs of this kind. The policy-specification language is discussed in more detail in Section 3.3.

Credentials can be a location the actor needs to be at, an identity the actor needs to have, or data the actor needs to possess.

Processes are used to define the dynamics in the model. For example, exchanging data between interconnected devices is modelled with the help of processes.

Our model is closely related to ExASyM [PH08] and Portunes [DPH10a], and therefore also to Klaim [BLP02; NFP98; GP03]. While in Klaim process mobility is modelled by processes moving from node to node, in our approach processes reside in special nodes that move around with the process. We choose this approach to make the modelling of (movement of) actors and items carried by actors more intuitive and natural. For completeness we give the syntax and semantics of our approach in Section 4.1 and Section 4.3 respectively.

Our model is expressive enough to model containment of items. Containment represents for example the fact that a workstation contains a hard drive that contains a file. In the formal language we use (described in the next chapter) we would represent the workstation as an item with a location; this location in turn would contain an item representing the hard drive; this item's location would contain data representing the (intangible) file.

We interpret containment as being transitive: if item a contains item b, and item b contains the data d, then we say a contains d transitively, and b contains d directly.

3.3 Policies

Policies are an important component of the socio-technical system model presented in the previous section. Policies play two roles. On one hand, in our model we support two general mechanisms for modelling and enforcing security policies, namely local and global policies. On the other hand, in addition to the modelling, both local and global policies are at the core of the attack generation presented in Chapter 5.

In the model, *local policies* are concerned with the actions of discrete entities, *e.g.*, single actors, and *global policies* are concerned with the overall system state. After discussing local policies in Section 3.3.1, we cover global policies in Section 3.3.2. Finally, we describe the syntax of our policy language in Section 3.3.3.

3.3.1 Local Policies

Local policies consist of a set of required credentials and a set of *actions* that are allowed when the required credentials are presented:

$$\begin{aligned} \mathit{LocalPolicies} &= \mathcal{P}\left(\mathit{RequiredCredentials} \times \mathit{Actions}\right) \\ \mathit{Actions} &= \mathcal{P}\left(\{\mathbf{in}, \mathbf{read}, \mathbf{out}, \mathbf{move}, \mathbf{eval}\}\right) \end{aligned}$$

The actions come from the set of actions supported by the modelling formalism acKlaim, presented in Chapter 4.

To ease the presentation, we formalise credentials as terms from the term algebra over a suitable signature, yielding a flexible and expressive, yet simple, formalisation. The signature is chosen based on a concrete system model, and contains enough structure to represent the model's important features. In our running example, we would expect the signature to at least contain such elements as cards, pin codes, locations, accounts, and actor ids.

Since signatures are highly model-dependent, we merely assume the existence of relevant signatures, $\Sigma = (S_\Sigma, F_\Sigma)$, for the models we consider, where S_Σ is the set of sorts and F_Σ is the set of function symbols over the sorts in S_Σ. We will not go into more details with defining signatures and term algebras here, but refer instead to the comprehensive treatment by [MT92] (also, see below for an example).

Thus, we can now give the formal definition of required credentials:

$$\mathit{RequiredCredentials} = \mathcal{P}\left(T\left(\Sigma, \mathit{Vars}\right)\right)$$

where $T(\Sigma, \mathit{Vars})$ is the term algebra generated by the signature Σ and variables found in Vars. Variables are needed to express generic or parameterised policies.

An example for a local policy is the following:

$$(\{\mathit{card}(\mathit{pin}(X)), \mathit{pin}(X)\} : \{\mathbf{in}\})$$

It expresses that one needs a card with a PIN code, and the corresponding PIN code for this card in order to be allowed to perform the **in** action at the protected

location. The set of credentials $\{card(pin(X)), pin(X)\}$ represents the formal way of the requirement that the actor should present a valid card (expressed by $card()$), containing a PIN code (expressed by $pin(X)$, where the variable X denotes that the exact PIN code itself is irrelevant as long as there is one), and should provide the relevant PIN code for this card (expressed by $pin(X)$, where the variable X is used to link back to the PIN code on the card).

The simplest possible signature for realising this local policy consists of a single sort $\star$ and the function symbols $pin()$ and $card()$ as well as the natural numbers as constants (represented as 0-ary function symbols):

$$S_\Sigma = \{\star\} \qquad \text{and} \qquad F_\Sigma = \{\texttt{card}\colon \star \to \star, \texttt{pin}\colon \star \to \star, n\colon \star\}$$

where n is a natural number.

This formalisation leaves the obvious question of how to determine if a set of credentials presented by an actor is sufficient to meet the requirements of a given policy. We start by formalising that the set of concrete credentials, *i.e.*, credentials that can be provided by an actor, must be represented as a set of ground terms from the term algebra that contain no variables:

$$\mathit{ProvidedCredentials} = \mathcal{P}\left(T\left(\Sigma, \emptyset\right)\right)$$

With this definition, we can use first order unification, as defined by Robinson [Rob65], to determine if a set $C \in \mathit{ProvidedCredentials}$ of (concrete) credentials is valid with respect to a given set $R \in \mathit{RequiredCredentials}$ of required credentials: if C and R can be successfully unified (*i.e.,* there exists a substitution $\sigma \in \mathsf{Subst} : \sigma R \subseteq C$, where $\mathsf{Subst} = \mathsf{Vars} \to T(\Sigma, \emptyset)$ is the domain of all substitutions from variables to ground terms), then the credentials C are sufficient to satisfy the required credentials R of a given policy.

In our example we assume $R = \{card(pin(X)), pin(X)\}$. Using the unification-based resolution, we can now determine whether the provided credentials $C = \{card(pin(42)), pin(42)\}$ would suffice to get permission to perform the action **in** at the location being protected. It is not difficult to see that the substitution $\sigma = [X \mapsto 42]$ applied to R yields $\sigma R = \{card(pin(42)), pin(42)\} \subseteq C$ and thus the action **in** is enabled.

The system model also supports predicates in credentials. Predicates are used to establish facts about actors; in the example a predicate *isEmployee* could express that the actor is an employee of the service provider, and *isCustomer* could express that the actor is a customer of the company. Predicates are specified in the model, and become part of the knowledge-base used in unification, and consequently the term algebra.

3.3.2 Global Policies

Global policies express system-wide policies that must be enforced everywhere, at any time, in the system, *i.e.*, they must hold in all states of the model. A global policy describes a state or actions which are disallowed in the system, and are expected to be enforced system-wide. While in practice there will be many global policies, we assume without loss of generality that only one such policy exists; for several policies our attack tree generation approach would result in individual attack trees for each of the policies, and combine them with disjunction nodes in the final attack tree. We assume two basic kinds of organisational policies:

- Action-based global policies, which forbid actors to perform certain actions; and
- Location-based global policies, which forbid data to reach certain locations.

Action-based global policies are specified like local policies with required credentials and a set of actions; they also contain an additional component identifying the attacker, who can be an actor or a variable:

$$\mathit{GlobalActionPolicies} = (\mathit{Actors} \cup \mathit{Vars}) \times \mathit{RequiredCredentials} \times \mathit{Actions}$$

Of course, the set of actions here specifies the prohibited actions. Location-based global policies are considerably simpler, since they only specify an asset and a location:

$$\mathit{GlobalLocationPolicies} = \mathit{Asset} \times \mathit{Location}$$

In our model, the location-based global policies are a specialisation of action-based global policies; for data to reach a location it either must be co-located with an actor, who must have input the data, or it must have been output at that location, which in turn again requires that an actor has input the data. These policies can therefore be translated to an action-based global policy that forbids inputing the data in question.

An example for an action-based global policy could forbid employees to reach a certain location: $(\{X\}, \{\mathit{isEmployee}(X)\}, \{\mathbf{move}(l)\})$. An example for a location-based global policy could specify that a file containing credential data must not leave the company's premises: $(\{\mathit{fileX}\}, \{\mathit{Outside}\})$.

3.3.3 Syntax Specification

Figure 3.2 presents the syntax of our policy specification language for both local and global policies.

Local policies consist of pairs of credentials and actions. The semantics of these pairs is that an actor must provide all the credentials in order to be allowed to perform the respective actions. It is possible that one policy definition consists of several pairs of credentials and actions. Each pair would then represent an alternative policy.

Global policies specify an actor or a variable, which constrain whom the policy should apply to, a set of credentials, and a set of actions, that actors are supposed *not* to perform.

Credentials are represented by a set possibly containing items, data, identity, location, or predicates. An item has an id and can contain other items or data, and data has an id and must either have a value, or be bound to a variable. An identity is matched against the actor's identity, a location specifies where the actor must be located to perform an action, and predicates support specification of properties of actors.

Variables provide a further means to tie required credentials to provided credentials, as well as policies to processes. We discuss the latter in the dedicated to processes Section 3.4. It should be noted that variables are introduced for convenience; for finite models, one can just instantiate all variables with all possible values in the policy definitions.

An enabled action can also take arguments, that either are values or variables that are bound to values from the credentials that have enabled the action. The **eval** action executes a process **P** with some arguments, and the **move** action permits the actor to move to the location.

3.4 Processes

The notion of processes has been introduced to model the movement and exchange of data in the model. Similarly to the actors in the model, who can move data with them from one location to another along the edges in the physical domain, data can move along the edges in the digital domain through the processes. In other words processes govern the rules on how the data is handled

$$
\begin{array}{rcl}
\mathit{LocalPolicies} & := & (\ \mathit{credentials}\ :\ \mathit{actions}\ )\star \\
\mathit{GlobalPolicies} & := & (\ (\ \textbf{actorID} \mid \textbf{variable}\ )\ ,\ \mathit{credentials}\ ,\ \mathit{actions}\ )\star \\
\mathit{credentials} & := & \emptyset \mid \{\ (\ \mathit{item} \mid \mathit{data} \mid \mathit{location} \mid \textbf{actorID} \mid \textbf{pred (}\ \mathit{argument}\star\ \textbf{)}\ )\star\ \} \\
\mathit{item} & := & \textbf{id}\ (\ (\ (\ \mathit{item} \mid \mathit{data}\ )\ )\ )? \\
\mathit{data} & := & \textbf{id (}\ \textbf{value} \mid \textbf{variable}\ \textbf{)} \\
\mathit{actions} & := & \emptyset \mid \{\ (\ \mathit{in} \mid \mathit{out} \mid \mathit{move} \mid \mathit{eval}\ )\star\ \} \\
\mathit{in} & := & \textbf{in}\ (\ (\ \mathit{argument}\star\ \textbf{)}\ )? \\
\mathit{out} & := & \textbf{out}\ (\ (\ \mathit{argument}\star\ \textbf{)}\ )? \\
\mathit{eval} & := & \textbf{eval}\ (\ \textbf{P (}\ \mathit{argument}\star\ \textbf{)}\ )? \\
\mathit{move} & := & \textbf{move} \\
\mathit{argument} & := & \textbf{value} \mid \textbf{variable} \mid \_
\end{array}
$$

Figure 3.2: The syntax of our policy language.

in these locations.

Processes are closely related to the notion of a *tuple space*, residing at each location. A tuple space is a set of assets (data or items) and their corresponding values. Processes, running at each location, perform various operations on a tuple space relating to their own location, or some other location. Examples of operations include writing data into a tuple space and reading data from it (destructive and non-destructive read operation). Policies at corresponding locations govern which of these tuple space operations are permitted, and processes themselves govern the way in which the permitted tuple space operations handle data.

Formally processes handle tuples of data – a set of parameters and their corresponding values without any particular structure or pre-defined values. The tuple in the input process filters all the output requests at the current locations and subsequent processes handle the way in which the tuple concerned is handled at the location – it may be stored in the local tuple space, or a subsequent action `out` may be launched to perform a write operation into a tuple space of another location.

A process is typically a sequence of several sub-processes. The first sub-process

is, as a rule, the input sub-process, which acts as a filter, filtering the tuple space operations of interest. Other sub-processes in a sequence may be related to the input sub-process by a sequential or parallel execution operator. For instance, a process definition of `in` A `.out` B means that tuple space operations, matching the credential A would match the input filtering rule. The sequential operator dot (.) means that the subsequent output process will be applied to the data received by the processes which were permitted to input some data into the tuple space of a considered location. The output process outputs some data B to the destination.

The enabling factor for tying policies and processes together, is the action **out**: it enables actors with the right credentials to output a token in the tuple space of the location protected by this policy. At the same location, one or more processes can wait for "their" token by executing an action **in**, which blocks until the expected input becomes available. In our running example, which is presented in detail in Section 3.6, the policy at the bank account

$$(\{C\}:\{\mathbf{out}(\text{“}transfer\text{”},,,,)\})$$

is tied to the process running at the bank computer (P_C). When the credentials of a certain account are provided, the process initiating the money transfer is triggered.

in $((\text{“}transfer\text{”}, !amount, !myAccount, !myAccountCredentials, !receiver))\ @C.$
read $((\text{“}credentials\text{”}, myAccountCredentials))\ @myAccount.$
read $((\text{“}balance\text{”}, !balance))\ @myAccount.$
out $((\text{“}balance\text{”}, balance - amount))\ @myAccount.$
in $(\text{“}balance\text{”}, !balance)\ @receiver.$
out $(\text{“}balance\text{”}, balance + amount)\ @receiver$

It should be noted that this approach also can model non-deterministic systems by adding two or more processes that all wait for the same token.

3.5 The Human Component

In this section we describe work we have done in an attempt to express the human component in our model. In Section 3.5.1 we illustrate restructuring of our model so we can define the human behaviour as an independent component, making the model parametric with respect to the behaviour. This would enable the possibility to "plug-in" different kinds of behaviour and thus overcome the restrictions of the existing modelling approaches. In Section 3.5.2 we present another attempt - this time aiming to model human behaviour, by relating it to a fundamental model in sociology, using higher order logic.

As we will see towards the end of the chapter, the research presented in this dissertation has evolved to a level, where light is shed onto the human behaviour component only after the attack trees have been generated from the model. For this reason, it also went out of the scope of our contribution to the TRE$_S$PASS project. Nevertheless, in the very last Chapter 7, for clarity reasons, we describe shortly the scope of our work in relation to the other project parts and discuss, among others, how they handle the human behaviour component in terms of defining attacker profiles.

3.5.1 Externalising Behaviour

Here we introduce an approach to a new, with respect to the literature, modular model structure. It is to some extent inspired by some of the already existing formalisms mentioned in Chapter 2, but in addition it aims at providing a more straightforward and structured way of modelling human behaviour and including it in the analysis.

A fundamental change to the model structure supported in existing models is the explicit addition of a component deciding the behaviour of actors. This change can go along with a further restructuring that also introduces separate components representing infrastructure, data, and actors, respectively.

An interesting aspect to investigate is the effect of *mixed behaviours*, for example, an attacker minimising the risk of detection until the attack has succeeded or the attacker has been detected, and then minimising the time needed to leave the organisation.

In the following subsections we describe the work in more detail. As at the time of carrying out the research described in this section the model described in this dissertation was ongoing work in early stage, we applied the approach to ExASyM. It is important to note that the framework is independent from the underlying model, and can be added to any of the models discussed in the related work, as well as to other system models.

3.5.1.1 Modular Structure

The biggest difference compared to the layout of the existing system models and analysis is exactly this externalisation of behaviour. While in their current version the different models all deeply integrate the behaviour into the analysis, and make it hard or impossible to change the behaviour, we extend the model

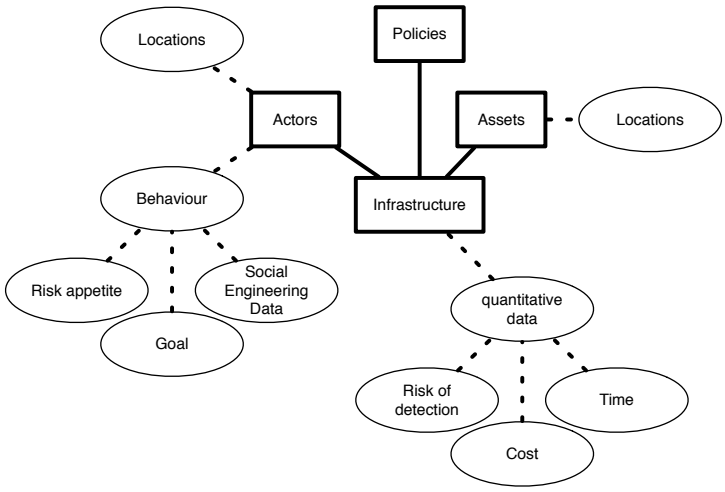


Figure 3.3: The modular system model structure. The explicit behaviour component and the quantitative data extend the core consisting of infrastructure, assets, policies, and actors. Each actor has a separate behaviour that is queried whenever the analysis or simulation needs to perform an action. The data associated with the behaviour then determines, which action will be performed. The *goal* describes, which mode the actor is in, *e.g.*, minimising risk of detection or time for actions. Both actors and assets are associated with locations that describe where they are located in the model.

with a behaviour component, which determines the actors' behaviour whenever queried by the system model.

Figure 3.3 shows the explicit structure of the framework and the interaction between the components. The behaviour component takes as input the actor, the actor's location, the data known, and the state of the system, and returns which action the actor decides to perform. Each actor has a separate behaviour that is queried whenever the analysis or simulation needs to perform an action. The *goal* describes, which mode the actor is in, *e.g.*, minimising risk of detection or time for actions.

Models with such an explicit behaviour component ease experiments and analyses of systems with different kind of attackers, simply by replacing the behaviour. It becomes straightforward, for example, to model actors who are opportunists, who just want to obtain the item of interest and get out again, or to model

behaviour driven by resource constraint such as time or risk of detection. Also the behaviour attributed to insiders, namely that they tend to repeat actions as long as they are not caught [FH02], and that they become less risk-aware over time, can easily be modelled.

3.5.1.2 Quantitative Data

The other extension over existing models, and also closely related to the addition of a behaviour component, is the introduction of quantitative properties for entities such as actors, actions, and locations. Supporting quantities in the model opens opportunities for new classes of questions one can ask the model. Depending on the quantities chosen, it would allow not only to ask whether a possibility of an attack exists, but also to get estimation about, for example, the cheapest attack in terms of cost, time, or risk of detection. We believe that this extension is a step closer to a more realistic approach to system analysis for vulnerabilities, as it also enables the modeller to use parametrisation of the model with behaviour of actors.

In a first step we add a set of "straightforward" metrics to the models, including:

- For actions the time to perform this action, the risk of detection when performing it, and the cost of performing it;
- For actors the likelihood of a social engineering attack to be successful and the risk appetite of the actor; and
- For locations the risk of detection at this location (for example due to surveillance cameras).

It is important to note that most of these annotations depend on some kind of context. The time it takes to perform a certain action certainly depends on the actor, on the location, on the risk appetite, and many other factors.

As it is out of our scope, we do not consider where this quantitative information comes from. There are existing studies that analyse insider behaviour, and these can be used to provide data for the models, as could be sociological analyses and surveys.

3.5.1.3 Kinds of Behaviour

The new behaviour component eases experiments with different kinds of behaviours, and also with phase shifts between different behaviour within the same analysis.

Analysing Processes. One of our first approaches to identifying insider threats in system models was the analysis of a given process describing actions of an actor in the modelled organisation [PHN07b]. This can be compared to static analysis of programs; while this approach makes sense in the case of programs, it is unclear how one would obtain the necessary process representing a complete trace of actions of an actor performed during an attack.

Representing and analysing this "predetermined" behaviour in the modularly structured model is straightforward. The behaviour is instantiated with the process P that one wants to analyse, and every time the model queries for the next action of the actor, the next action in the process P is returned. In this scenario, the behaviour does completely ignore the state of the system and the location of the actor when picking the next action.

The Greedy Explorer. When moving away from processes, the static analysis approximates all possible actions performed by actors in the system. To do so, the analysis must assume that every action that can be performed in the model also will be performed [PH08]. Also this scenario is easily re-implemented using the external behaviour component. Whenever the model asks the actor for the next action, the actor gets the location where he is currently located, obtains all actions permitted at this location by the access control policy, and iteratively returns the admissible actions.

More Advanced Approaches. The real benefit of the modular behaviour component becomes apparent when considering new behaviours such as mixed behaviour with different phases, as discussed above, or history-based behaviour, which takes previous actions into account. All these could have been introduced also in the original model, but at the cost of changing the behaviour of all actors.

```
1: equivalent()
2: /* perform (log-)equivalent actions */
3: changed = true
4: while changed do
5:   changed = false
6:   for all actors n do
7:     for all locations l that n might be located at do
8:       for all locations l' reachable from l in one step do
9:         simulate all actions that n can perform on l'
10:        for each action set changed if n at location l learns a new data item
11:      end for
12:    end for
13:  end for
14: end while
```

Figure 3.4: The greedy explorer algorithm presented in [PH08]. For each actor in the system we check for all locations he can be located at whether he can perform any actions, and if yes, perform them. When switching to the external behaviour component, the only change necessary is in line 9, where we query the behaviour for actor n at location l for actions that can be performed at l'.

3.5.1.4 Restructuring a System Model

In our work we have used the structure introduced in the previous section to obtain a modular system model, in which the behaviour of actors is a parameter of the model. The model described here has been implemented in [Thy13] as a restructuring of the original model.

The original algorithm is shown in Figure 3.4 and shows the interweaving of the behaviour with the analysis. Introducing a separate behaviour component induces almost no changes in the existing analyses. At the same time it helps separating the core of the analysis (at which locations do we check for actions to perform) from the actual decision, which actions will be performed. This is an immediate result of the modularisation of the system model. Another side effect is that we can simulate actors with *different* behaviours.

After adding a behaviour component to the system, the analyses available had to be adapted accordingly. One of the main goals of the restructuring was to ease the addition of new analyses, and to provide APIs to support access to the model's functionality, including the behaviour. The work in [Thy13] showed that the adaptation of the existing analyses in the new model was straightforward.

```
1: for all actors n do
2:   for all actions a in behaviour(n) do
3:     simulate action a
4:     for each action set changed if n at location l learns a new data item
5:   end for
6: end for
```

Figure 3.5: The reimplementation of the loop in the pseudo-code in Figure 3.4 (lines $6-13$) using the behaviour component. It should be noted that all the information regarding, *e.g.*, possible locations of the actor is stored in its state, so the behaviour does not need additional input.

3.5.1.5 Lessons Learned

The addition of an external behaviour component to ExASyM, and the reimplementation of existing analyses as well as new analyses was surprisingly easy. The code for existing analyses became much clearer, since the representations of actors and data contain all their locations, so it is the behaviour component that decides what to do in case of an actor possibly being located in different locations, not the analysis. This in itself makes adding the behaviour component a worthwhile task.

Also for developing new analyses the encapsulation of decisions about possible actions into the behaviour is beneficial, as the resulting analyses concentrate on computing the analysis result, and do not have to keep track of where actors and data might be located.

3.5.2 Modelling Human Behaviour with Higher Order Logic

In collaboration with members of the computer science group at Middlesex University, London, UK, we have done some work in the direction of formalising our approach in Isabelle [NPW02]. In Appendix C we show the Isabelle theory for a small subset of the model reflecting the running example.

We related the human component in socio-technical systems to a fundamental model in sociology suggested by one of the forefathers of sociology, Max Weber [Web78]. His basic process of sociology, *i.e.*, "understanding explanation" includes three steps:

- "interpreting understanding" is the step, where the sociologists try to understand how the actors interpret their situation,
- the actor's subjectively meaningful action, and
- the effects of the action itself

An approach following the logic of explanation [HO48], which maps to Weber's three steps, is used for explaining sociological phenomena. This approach introduces a view on actors and by doing so Weber's three steps translate to a macro-micro-macro-transition. In other words, to explain the global phenomena, the global facts from the macro level are broken down into local views of the individual actors (representing the micro level), and then the steps from the micro level are generalised and brought up to the macro level.

Below we briefly discuss how the three transitions are formalised and what is their relation to security modelling. Then we shortly present how we address each of the steps using Higher Order Logic (HOL).

Macro to Micro In this step a *situational logic* maps the global environment onto the actor's view.
Usually, formal security models do not model the situational logic. The attacker skills may be quantified, but neither the motivation nor the social or psychological factors are addressed.
The theory supporting this situational logic needs to present the attacker's mental characteristics. A framework for characterising insider threats, which proposes insider threads taxonomy [Nur+14], identify different classes of such characteristics. Modelling them in HOL is rather easy, simply by using the HOL datatype.

Micro to Micro The individual actor's view is represented by a *logic of selection*, which describes how the actor chooses his actions based on the situation and his perception. This choice, for example, could be expressed in purely normative way - the actor's decision is the result from given rules based on predetermined norms.
The attacker's capabilities and the characterisation of his/her actions is addressed in several formal security models, for example in protocol verification. In many security models, the behaviour of the attacker is based on the Dolev-Yao model. As mentioned earlier, such kind of attacker is the most powerful kind of attacker.
In the micro level we are interested in modelling the actors within their environment. The environment is modelled to represent the physical and technical layer

mentioned in Section 2.1.2. Policies express prerequisites in terms of conditions, defined by predicates, and a set of enabled actions, granted if the prerequisites are fulfilled.

Micro to Macro In the third step, the micro-sociological results are lifted back to the macro level by an *aggregation logic*. The result then would be the explanation of the sociological phenomenon.
The aggregation logic is a constant factor in security modelling, as security is the only sociological aspect we are interested in. The economical aggregation of attacks though is of utmost interest, *i.e.*, estimation of the direct and implicit damages of the attacks.
The effect is that of violating a policy, therefore the effect to the macro-level is subsumed by the negated global policy. For example, the effect of the attack is, that classified data reaches the outside of the corporate network.

Using the approach described in this section, once a scenario is modelled, a given attack is then proved as an Isabelle/HOL theorem.

3.6 Modelling the Running Example

In this section we illustrate how the modelling methods described earlier in this section are applied to the running example presented in Section 3.1. To ease the reader, we first present a graphical overview of the use case scenario. Using the model described earlier in this section, the relevant locations, actors, and assets in the case study become nodes in a graph shown in Figure 3.6:

- Alice, an elderly person, and Charlie, another actor,
- Alice's home including a door that controls access, a television with a set-top box, and a remote control,
- Alice's payment card, which contains the PIN and the name of the card's owner, the PIN itself, which is known by Alice, and her password for her bank account,
- Charlie's payment card, which similarly contains the PIN and the owner's name, as well as the PIN itself, which Charlie knows, and
- an ATM, a bank, and a bank computer, that represents the payment services.

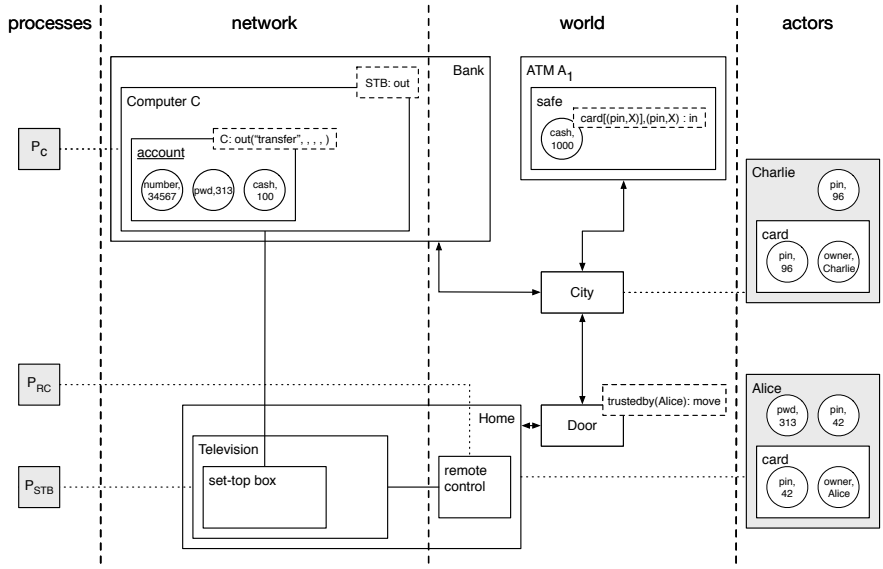


Figure 3.6: Graphical representation of the running example. The white rectangles represent locations, the big grey rectangles represent actors and contain the assets known or owned by the actor, the round nodes represent assets, and the small squares represent process nodes. Solid lines represent the physical connections between locations. The dashed rectangles in the upper right part of some nodes represent the policies assigned to these nodes.

Access to assets is governed by policies as described above; for example the card and the PIN are needed to access the money in the ATM, the remote control is needed to start a process for transferring money from the set-top box, and the role of an IPTV technician is needed to replace the firmware on the set-top box.

There are two primary actors: Alice, an elderly lady, having a role of a customer, *i.e.*, receiving care taking services, and Charlie - a care taker, *i.e.*, having a role of an employee.

In Figure 3.6, for example, the node representing the actor Alice has a pin code and a card. The card in turn contains information about the owner and the pin code for the card.

As described in Section 3.3.1, generic policies may contain variables that are bound on their first occurrence in the unification, and checked afterwards. In the

evaluation example shown in Figure 3.6 , the policy for the money in the ATM is $(\{card(pin(X)), pin(X)\} : \{\mathbf{in}\})$. To fullfil this policy, an actor is required to present a card, which contains a pin code, as well as the pin code matching the pin code on the card. Doing so, and applying unification (as explained above) to the concrete credentials $\{card((pin, 1234)), (pin, 1234)\}$, results in a successful unification, where $X = 1234$, and thereby enables the action **in**. In the example this action would represent that the actor is allowed to get money.

The small squares on the left side in Figure 3.6 represent the processes at the remote control ($\mathrm{P_{RC}}$), the set-top box ($\mathrm{P_{STB}}$) and the bank computer ($\mathrm{P_C}$), respectively. Their definitions can be seen below:

$$P_U := \mathbf{out}\,((\text{“}pin\text{”}, pin))\,@RC$$

$$\begin{aligned} P_{RC} := \; & \mathbf{in}\,((\text{“}pin\text{”}, !pin))\,@RC. \\ & \mathbf{read}\,((\text{“}pin\text{”}, pin))\,@card. \\ & \mathbf{out}\,((\text{“}pincheck\text{”}, \text{“}ok\text{”}))\,@STB \end{aligned}$$

$$\begin{aligned} P_{STB} := \; & \mathbf{in}\,((\text{“}amount\text{”}, !amount))\,@STB. \\ & \mathbf{in}\,((\text{“}receiver\text{”}, !receiver))\,@STB. \\ & \mathbf{in}\,((\text{“}pincheck\text{”}, \text{“}ok\text{”}))\,@STB. \\ & \mathbf{out}\,((\text{“}transfer\text{”}, amount, myAccount, myAccountCredentials, receiver))\,@C \end{aligned}$$

$$\begin{aligned} P_C := \; & \mathbf{in}\,((\text{“}transfer\text{”}, !amount, !myAccount, !myAccountCredentials, !receiver))\,@C. \\ & \mathbf{read}\,((\text{“}credentials\text{”}, myAccountCredentials))\,@myAccount. \\ & \mathbf{read}\,((\text{“}balance\text{”}, !balance))\,@myAccount. \\ & \mathbf{out}\,((\text{“}balance\text{”}, balance - amount))\,@myAccount. \\ & \mathbf{in}\,(\text{“}balance\text{”}, !balance)\,@receiver. \\ & \mathbf{out}\,(\text{“}balance\text{”}, balance + amount)\,@receiver \end{aligned}$$

The user (presented as a simple process P_U) provides his or her pin code at the remote control. The process running at the remote control P_{RC} inputs the pin from the remote control. The remote control then outputs the pin at the set-top box.

The process P_{STB} initiates a money transfer at the set-top box in Alice's home. After inputting the amount and the receiver's account number, the user's pin

is required to be provided at the remote control. After input of the pin, it then outputs the information at the bank computer.

The process P_C at the bank computer inputs the amount to be transferred, the sender account number, together with the sender's credentials, as well as the receiver's account. After that the process reads the sender's credentials as well as his balance. The process then outputs the new balance of the sender and reads the balance of the receiver. Finally, the process outputs the updated balance to the receiver's account.

In the example from Figure 3.6 , the global action-based policy could be

$$not(\{X\}, \{isEmployee(X), card[(owner, Y)], isCustomer(Y)\}, \{\mathbf{in}\})$$

stating that an actor X is not allowed to use a card as credential when performing an action **in**, if the predicate *isEmployee* is true for X and the card is owned by an actor Y, for whom the predicate *isCustomer* holds. In the example, the only possible binding for X is Charlie, and the only possible binding for Y is Alice, and the action **in** would represent obtaining money at an ATM.

The underlying formalism of our approach is described in details in the next Chapter 4.

3.7 Implementation

We implemented our model into a prototype tool written in Scala, that takes as an input an XML file. The XML schema can be found in Appendix A. The input is then parsed into the knowledge base, which is used for the further analysis of the model, *e.g.*, the attack generation described in Section 5.2. Figure 3.7 shows the architecture of the final tool.

3.8 Concluding Remarks

In this chapter we have outlined the system model components, we have defined the policy specification language, and we have also described our attempts to tackle the problem of modelling human behaviour. Without going into details of the underlying process calculus, which is described in the following chapter, we have presented our system model on the running example.

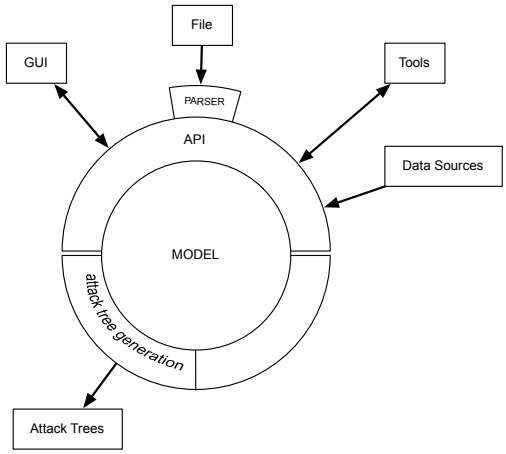


Figure 3.7: Overview of the tool. The current prototype implements the parser to read input files, and the attack tree generation. Direct interfaces to analyses, data sources, and a GUI, and probably also further interfaces, are part of the future work.

With regards to the policy-specification language, it is worth the remark that our local policies correspond to automated security policies in Sterne's policy framework presented in Section 2.1.3. At the same time, our global policies match both security policy objectives and organisational security policies.

Moreover, the notion of credentials enables the possibility of integration with other policy-specification languages. Term algebras are very flexible and an easy to use approach for the syntax of credentials. It is also straightforward to encode/convert the syntax of the policy-specification languages mentioned in Section 2.4 into term algebra notation.

At the end, the human factor in the model is reflected in one single way. The attacker in our model is quite powerful, though not exactly the Dolev-Yao type of attacker. Despite of the fact, that the attacker knows where and how to obtain needed credentials, initially they are not necessarily in his possession as opposed to Dolev-Yao attacker, who knows everything right from the very beginning.

Initially, as described in Section 3.5, we explored different possibilities to define the human behaviour as part of the model. With the model evolving within the big picture of the TRE$_S$PASS project, we realised, that it makes the model unnecessarily too complicated. The analysis of the model is not affected of the

behaviour itself. Its role starts playing when the attack vectors are generated. The notion of behaviour is expressed by defining the so called *attacker profiles* - a profile defining either an individual or a group with the help of the roles, which the actors have assigned in the model. A given attack vector might not be possible given the attacker profile, but it is still a valid attack vector derived from the analysis of the model. For better clarity, we describe in more details the major scopes covered in the TRESPASS project in Section 7.1.

The need for quantified data in the model is tightly connected to the behaviour. Therefore, assigning quantified data on the leaves of the attack trees can then represent, among others, probability of success, cost or time distribution with respect to performing the action. In our work, we do not consider where this quantities come from.

Chapter 4

The Process Calculus

In the previous chapters, we have presented the system model components, we have described the policy-specification language and we have demonstrated how to apply our modelling framework to an IPTV case study. Now, since we want to generate attack trees analytically from the model, we need to define an underlying formalisation of our model, which will enable the analysis for the generation. In this chapter we present the process calculus we use.

Process calculi are a well-established formal modelling technique, well-suited to describe complex behaviour in terms of relatively simple interacting components. Last but not least, they lend themselves to develop verification tools like, for example, the one we present in Chapter 5 for generating attack trees.

We formalise our model using a variation of the Klaim language [NFP98]. We start with presenting the syntax in Section 4.1. We proceed with a discussion on the reference monitors, used to model the policy-checking mechanism, in Section 4.2, before moving on to describing the semantics in Section 4.3.

4.1 Syntax

The syntax of our calculus is shown in Figure 4.1. Since it is very close to Klaim [NFP98], we first give a very short summary of the main features of Klaim based on [DGP05] and then point out the main differences that we introduce here.

In Klaim, nets are collections of *nodes* N that contain *processes* P and *tuple spaces* TS. A tuple space provides a repository of tuples that can be accessed concurrently. A node $n \in N$ is a pair $l :: C$, where locality l is the name of node n and C is the distributed component located at node n. We usually refer to a node n with its name l. Components C can be processes or tuples. A tuple t is a sequence of values. Tuples are inactive components that have either been there already in the initial configuration or have been output by a process during a computation at this location. The tuple space located at node l results from the parallel composition of all located tuples positioned at l. Pattern matching is used to select tuples from a tuple space where the syntax $!x$ is used to bind variables to values. For example, $match(!x, l)$ produces the substitution $\sigma = [l/x]$. An evaluated tuple et is a tuple which has only values, *i.e.*, it does not contain any variables. In Klaim, a template matches a tuple if both have the same number of fields. In our process calculus, both tuples and templates are of length two. Corresponding fields match, if they have the same name, or one is a variable and the other one a name.

Processes P can be built from the inert process **nil** and basic actions for read, write, move, execute, and creation of new nodes. For example, the action $\mathbf{eval}(P)@l$ sends process P for execution to node l. The action $\mathbf{out}(t)@l$ writes a tuple in the tuple space at node l.The action $\mathbf{in}(T)@l$ looks for a matching tuple et in the tuple space located at l. Templates T help the processes to be able to choose which tuples to input or read. They represent a format, which acts as a filter – the tuples should match the template. If et is found, the formal fields of T are replaced in the continuation process with the corresponding values of et. We consider in our adaptation of Klaim the actions $\mathbf{out}, \mathbf{in}, \mathbf{read}, \mathbf{eval}, \mathbf{move}$ for output, input, read, evaluation of processes, and moving of actors. A process can be a combination of several sub-processes constructed by parallel composition or it can be an invocation through a place-holder variable explicitly defined by an equation.

A location is a tuple space (which can be interpreted as a local database), where assets are stored. We divide localities into those representing infrastructure of any kind (l_i), such as rooms or work stations, and those representing actors (l_a). We omit the index when both kinds of localities can be used.

ℓ	::=	l_i	infrastructure location	N	::=	$l_a ::^{\delta} P$	actor node
	|	l_a	actor location		|	$l ::^{\delta} TS$	located tuple
	|	u	locality variable		|	$N_1 \parallel N_2$	net composition
P	::=	**nil**	null process	a	::=	**out** $(t)\,@\ell$	output
	|	$a.P$	action prefixing		|	**in** $(T)\,@\ell$	input
	|	$P_1 \mid P_2$	parallel composition		|	**read** $(T)\,@\ell$	read
	|	A	process invocation		|	**eval** $(A, \delta)\,@\ell$	remote evaluation
					|	**move** (ℓ)	move

TS	::=	$\emptyset \mid \langle et \rangle \mid TS, TS$	tuple space
T	::=	$(nameexpr, !x)$	templates
t	::=	$(nameexpr, valueexpr)$	tuple
et	::=	$(string, V)$	evaluated tuple
$nameexpr$	::=	$string \mid x$	name or variable
$valueexpr$	::=	$V \mid x$	expressions
V	::=	$string \mid Int \mid l$	values

Figure 4.1: The syntax of our calculus. We divide localities into those representing infrastructure of any kind (l_i), such as rooms or work stations, and those representing actors (l_a), or more generally processes. We omit the index when both kinds of localities can be used. Tuple spaces contain evaluated tuples. In contrast to Klaim we limit tuples (and templates) to pairs; the first component specifies the name, the second component is either a string (to represent knowledge) or a location (to represent items). Locations have a name (the l) and an access policy δ.

An asset can represent either data or items. Data in our calculus is also restricted compared to original Klaim. As we mentioned earlier, we only allow tuples and templates of length two – the first element is the name of the tuple and the second element contains its value, which can be a string or a location.

Data is stored as a pair – the first element is its name (string) and the second element is the value itself (string or an integer). Data represents information, which is "known" by an actor, for example, the pin code in Figure 3.6 known by Alice. Items are represented by a pair of a value and a location, for example a payment card, which in turn can contain further assets in their tuple space. An example for the latter is the information about the owner and the pin code contained in the chip of a payment card.

Locations in our calculus are divided into *infrastructure* locations l_i and *actor* locations l_a. As stated above, processes can only be executed at actor locations, and while in Klaim processes move around in the system, in our approach the node hosting the process moves around. We have chosen this approach mostly

to unify the interaction of the system with items carried by an actor and with items stored at a location. To integrate actors as moving nodes, we stipulate as a well-formedness condition that they are always defined as singleton nodes, *i.e.*, $\forall\, l \in N_a.\, |\{(l,x)|(l,x) \in E\}| = 1$. If this condition is initially true, it is preserved by the semantics of the actions, in particular, **move**.

An alternative to this approach would have been to store locations representing items in a key set at the process, similar to [PHN07b], but then the actions **in** and **out** would have required different semantics, depending on whether the goal location of the action is a location in the system (where the data would be stored in the tuple space) or an actor (where the data would be stored in the key set).

The other difference between our approach and Klaim is that locations have an access control policy δ attached, *e.g.*, $l ::^{\delta} P$, similarly to [De +10]. The policies δ regulate access to the location. While in general a policy could contain several pairs of required credentials and enabled actions, we assume for sake of simplicity that policies only contain one such pair.

4.2 Reference Monitors

Reference monitors are an approach to check the enforcement of a given security policy on a system. In other words, it is the reference monitors, which make authorisation decisions with regards to policies [D1.2.2], which have been discussed in Section 3.3. For the semantics we first introduce a reference monitor similar to [PHN07b], which can be seen in Figure 4.2. The function *grants* checks in the infrastructure $\mathcal{I}$ (defined below) whether an actor l is allowed to perform an action a at target location t. Computing *grants* checks the infrastructure to see whether l is located either at t or at a location next to t, and whether the actor is allowed to perform the action by the policy at t based on its location, its id, or its knowledge.

Actor *actor* is allowed to perform action a at target location t if *actor* is located at t or at a neighbouring node, and if the access control policy at t can be discharged by *actor*. This check is performed by function *enables* using unification of actor knowledge and required credentials. This reference monitor is used in all semantic rules, described in the following section, to decide whether the conclusion of the rule is fired or not. If the *grants* function returns false, the semantic rule blocks.

Both functions, *grants* and *enables*, are parametrized on the infrastructure $\mathcal{I}$,

$$grants_{\mathcal{I}} : \mathsf{N}_a \times \mathsf{N} \times \mathsf{Policies} \times \mathsf{Actions} \to \{\texttt{true}, \texttt{false}\}$$

$$grants_{\mathcal{I}}(actor, t, \delta_t, a) = \begin{cases} \texttt{true} & \text{if } \mathcal{I} = (\mathsf{N}_i, \mathsf{N}_a, \mathsf{E}) \text{ and } \exists l' \in \mathsf{N}_i : \\ & ((actor, l') \in \mathsf{E} \wedge (l', t) \in \mathsf{E}) \vee l' = t) \wedge \\ & \quad enables_{\mathcal{I}}(actor, a, \delta_t) \\ \texttt{false} & \text{otherwise} \end{cases}$$

$$enables_{\mathcal{I}} : \mathsf{N}_a \times \mathsf{Actions} \times \mathsf{Policies} \to \{\texttt{true}, \texttt{false}\}$$

$$enables_{\mathcal{I}}(actor, a, \delta) = \begin{cases} \texttt{true} & \text{if } \delta = \emptyset \\ \texttt{true} & \text{if } \delta = (cred, act) \wedge unify_{\mathcal{I}}(actor, cred) \wedge a \in act \\ \texttt{false} & \text{otherwise} \end{cases}$$

Figure 4.2: Function *grants* checks whether an actor node l is allowed to perform action a at location t. This is allowed if l is located at t or at a neighbouring node, and if the access control policy at t can be discharged by l. This check is performed by function *enables* using unification of actor knowledge and required credentials. Both functions are parametrized with the infrastructure $\mathcal{I}$, which defines the structure of the model.

which defines the structure of the model. The infrastructure $\mathcal{I} = (\mathsf{N}_i, \mathsf{N}_a, \mathsf{E})$ consists of sets of nodes representing the physical locations (N_i), actor and process locations (N_a), and a set of edges ($\mathsf{E} \subseteq \mathsf{N} \times \mathsf{N}_i$), where we define $\mathsf{N} = \mathsf{N}_i \cup \mathsf{N}_a$. This represents that actors can be located at infrastructure locations, and there are edges connecting the infrastructure locations. An edge $(actor, l) \in \mathsf{E}$ describes that actor *actor* is located at location l.

4.3 Semantics

In this section we present the semantics of our process calculus. After briefly discussing what the differences from the existing approaches are, we explain each of the rules presented in Figure 4.3.

The semantics of our calculus is based on that of ExASyM [PH08]. Similar to ExASyM, the semantics is based on an infrastructure representing the underlying infrastructure $\mathcal{I}$. The processes performing actions on $\mathcal{I}$ are constrained by the policies δ on locations, which are enforced by reference monitors (boxes in the rules) similar to [PHN07b]. The *grants* function takes as an input an actor, a location, a policy at that location, and an action and ensures that the actor is allowed to perform the action at the certain location. In case there is a policy assigned to the location from the input, a check ensures that the actor fulfils

$$\frac{[\![t]\!] = et \qquad \boxed{grants_{\mathcal{I}}(l, l', \delta', \mathbf{o})}}{\mathcal{I} \vdash l ::^{\delta} \mathbf{out}\,(t)\,@l'.P \parallel l' ::^{\delta'} TS \rightarrowtail \mathcal{I} \vdash l ::^{\delta} P \parallel l' ::^{\delta'} \langle et \rangle, TS}$$

$$\frac{match([\![T]\!], et) = \sigma \qquad \boxed{grants_{\mathcal{I}}(l, l', \delta', \mathbf{i})}}{\mathcal{I} \vdash l ::^{\delta} \mathbf{in}\,(T)\,@l'.P \parallel l' ::^{\delta'} \langle et \rangle, TS \rightarrowtail \mathcal{I} \vdash l ::^{\delta} P\sigma \parallel l' ::^{\delta'} TS}$$

$$\frac{match([\![T]\!], et) = \sigma \qquad \boxed{grants_{\mathcal{I}}(l, l', \delta', \mathbf{r})}}{\mathcal{I} \vdash l ::^{\delta} \mathbf{read}\,(T)\,@l'.P \parallel l' ::^{\delta'} \langle et \rangle, TS \rightarrowtail \mathcal{I} \vdash l ::^{\delta} P\sigma \parallel l' ::^{\delta'} \langle et \rangle, TS}$$

$$\frac{\mathcal{I}' = \left(\mathsf{N}_i, \mathsf{N}_a, \mathsf{E} \setminus \{(l,x) | (l,x) \in \mathsf{E}\} \cup \{(l, l')\}\right) \qquad \boxed{grants_{\mathcal{I}}(l, l', \delta', \mathbf{m})}}{\mathcal{I} \vdash l ::^{\delta} \mathbf{move}\,(l')\,.P \rightarrowtail \mathcal{I}' \vdash l ::^{\delta} P} \quad \mathcal{I} = (\mathsf{N}_i, \mathsf{N}_a, \mathsf{E})$$

$$\frac{\mathcal{I}' = \left(\mathsf{N}_i, \mathsf{N}_a \cup \{l_Q\}, \mathsf{E} \cup \{(l_Q, l')\}\right) \qquad \boxed{grants_{\mathcal{I}}(l, l', \delta', \mathbf{e})}}{\mathcal{I} \vdash l ::^{\delta} \mathbf{eval}\,(Q, \delta_Q)\,@l'.P \rightarrowtail \mathcal{I}' \vdash l ::^{\delta} P \parallel l_Q ::^{\delta_Q} Q} \quad \mathcal{I} = (\mathsf{N}_i, \mathsf{N}_a, \mathsf{E}), l_Q \notin \mathsf{N}_a$$

$$\frac{\mathcal{I} \vdash N_1 \rightarrowtail_{\mathcal{I}} \mathcal{I}' \vdash N_1'}{\mathcal{I} \vdash N_1 \parallel N_2 \rightarrowtail_{\mathcal{I}} \mathcal{I}' \vdash N_1' \parallel N_2} \qquad \frac{N \equiv N_1 \qquad \mathcal{I} \vdash N_1 \rightarrowtail_{\mathcal{I}} \mathcal{I}' \vdash N_2 \qquad N_2 \equiv N'}{\mathcal{I} \vdash N \rightarrowtail_{\mathcal{I}} \mathcal{I}' \vdash N'}$$

Figure 4.3: The semantics for the actions of our calculus. We assume that all locations exist in the infrastructure so we omit sanity checks. The action **out** outputs a tuple at the target location, the action **in** consumes a tuple, while the action **read** is non-destructive. In contrast to Klaim, a locality containing a process moves around together with the process. The action **move** reconnects the location of the process executing the action to the goal location, and the action **eval** creates a new location with the new process and connects it to the goal location.

the credentials in the policy and the action from the input is among the enabled actions *act* in the policy. The enabled actions are a subset of Actions, where

$$\mathsf{Actions} = \{\mathbf{i}, \mathbf{o}, \mathbf{r}, \mathbf{m}, \mathbf{e}\}$$

represent the simple actions from our syntax, where **i** stands for input, **o** stands for output, **r** stands for read, **m** stands for move, and **e** stands for eval.

The function then returns a Boolean value – *true* represents that the actor is allowed to perform the action at the specified location. Respectively, $false$ means the opposite, *i.e.*, that the action can not be performed. The formal definition of the *grants* function used in the reference monitors has been described in Section 4.2 (see also Figure 4.2).

$$match(V, V) = \epsilon \qquad match(!x, V) = [V/x]$$

$$\frac{match(T_1, et_1) = \sigma_1 \qquad match(T_2, et_2) = \sigma_2}{match((T_1, T_2), (et_1, et_2)) = \sigma_1 \circ \sigma_2}$$

Figure 4.4: Semantics for template matching.

The main difference to both ExASyM and Klaim is that the rules for **move** and **eval** take the infrastructure and actor nodes into account.

In the rules describing the basic actions in Figure 4.3, which are explained in detail below, $[\![t]\!]$ denotes the evaluation of a tuple, and $[\![T]\!]$ denotes the evaluation of a template. We also assume, that all locations in the infrastructure exist, therefore we omit sanity checks.

The action **out** places an evaluated tuple et in the tuple space of location l. It is worth noting, that the action **out** is asynchronous and always succeeds, as it operates on tuple spaces and those are said to be "always available for writing".

The action **in** attempts to input a tuple from the tuple space of node l. It takes a template T and tries to match it against all tuples at node l. The formalisation of the pattern matching can be seen in Figure 4.4. Whenever a matching tuple is found, it is removed from the tuple space of the node l. If no matching tuple is found, the action blocks. The execution of the rest of the process (if any) then continues, where the substitution σ, resulting from the template matching, is applied.

The action **read** is very similar to the action **in** with the only difference, that the evaluated tuple et is not removed from the tuple space. It is obvious that if a process can perform the action **in**, it will also succeed in performing the action **read**.

The rule for the action **move** removes any edges from the actor node ℓ_a to the infrastructure node ℓ_i, and adds a new edge to the goal of the action **move**. In other words, we simulate movement by changing the connectivity of the nodes.

The action **eval** takes a process Q and policies σ_Q as arguments, creates a new location ℓ_Q, and connects it to the action's target location ℓ'. Both actions **move** and **eval** result in a new infrastructure $\mathcal{I}'$.

Figure 4.4 shows the semantics for the template matching. In the first rule, if we match a value against itself, we produce the empty substitution, *i.e.*, the identity. The second rule matches a value against the defining occurrence of a variable and produces a substitution that binds the variable to the value. In the

$$
\begin{array}{rcl}
N_1 \parallel N_2 & \equiv & N_2 \parallel N_1 \\
(N_1 \parallel N_2) \parallel N_3 & \equiv & N_1 \parallel (N_2 \parallel N_3) \\
l ::^{\delta} P & \equiv & l ::^{\delta} (P \mid \mathbf{nil}) \\
l ::^{\delta} TS & \equiv & l ::^{\delta} (TS \mid \emptyset) \\
l ::^{\delta} A & \equiv & l ::^{\delta} P \qquad \text{if } A \triangleq P \\
l ::^{\delta} (P_1 \mid P_2) & \equiv & l ::^{\delta} P_1 \parallel l ::^{\delta} P_2
\end{array}
$$

Figure 4.5: Structural congruence on nets and processes.

third rule, provided that we have a substitution σ_1, resulting from matching the template T_1 and an evaluated tuple et_1, and we have another substitution σ_2, resulting from matching the template T_2 against an evaluated tuple et_2, then matching (T_1, T_2) against the pair (et_1, et_2) produces the composition of the two substitutions.

The structural congruence, denoted by the symbol $\equiv$, defines when two processes are congruent to each other ($P_1 \equiv P_2$). Figure 4.5 shows the rules for the structural congruence. The relation enforces that processes constitute a commutative monoid with respect to parallel composition and **nil**. We assume the symbol $\equiv$ to be a congruence over networks (and processes, respectively), *i.e.*, it has the reflexivity, transitivity and symmetry properties, as in standard Klaim.

4.4 Concluding Remarks

In this chapter we have seen the process calculus we use for our system model. We have presented its syntax, operational semantics, and the reference monitors used for checking the policies.

There is some tension between the need for a graphical notation for to be understandable to professionals and business users (on the one hand). On the other hand, we have the need of a formal language-based representation to enable the development of automated analysis techniques. It is of course highly desirable to establish a formal correspondence between the graphical model and the language-based model.

In the context of this dissertation, we limit to show examples in the graphical notation, that can be naturally translated into the process calculus presented in

this chapter. In the wider context of the TRE$_S$PASS project, an entire strand of research aims at producing tools for supporting graphical modelling, that enforce those constraints that make the graphical notation encodable into the language-based notation.

We do not expect the practitioners to fiddle with the process calculus. On the contrary, we would like to close the gap between the theoretical approach and the application-oriented users.

Chapter 5

Attack Generation

The manual identification of possible attacks on organisations is a creative process. However, it could be not only tedious, but also highly error prone. Moreover, in the case of complex enterprises, it is nearly impossible to adequately reflect changes in the structure of the organisation with respect to its risk assessment.

Since the interest of closing this gap, both within industry and academia, has been increasing significantly the last few years, there have recently been studies proposing automatic attack tree generation methods. However, none of them have addressed the deficiency in the classical risk assessment methodologies, namely the lack of social and human aspects in the attack steps. Using recent advances in system models, we have developed a technique to identify possible attacks analytically, *including* technical and human factors.

In this chapter we first present in Section 5.1 the attack generation technique based on graphical transformations, while sparing the reader any technical details concerned with the formalism of our system model. After that, in Section 5.2, we suggest the use of system models to systematically generate attack trees by invalidating policies.

The generated attack trees can then be used as an input to a traditional risk assessment process and thereby extend and support the brainstorming results.

We would like to note, that in both approaches we assume an implicit left to right order of the sub-goals in the attack trees.

5.1 High Level Graphical Transformation of System Models

In this section we present the graphical transformations of our attack generation technique. Since the transformations consider all relevant system components, the resulting attacks may include elements of human behaviour.

5.1.1 Transforming Models without Asset Mobility

In this section we consider assets in the modelled organisation to be immobile. This restriction, which will be lifted in the next section, simplifies the first presentation of transformations.

The transformation starts from the specification of an asset, which an attacker should not be able to obtain. The goal of the transformation is to generate an attack for every possible actor in the system after which that actor would have obtained the asset in question. The overall transformation is a generalised version of policy invalidation [KW13; KW14]:

1. Starting from the goal asset and the attacking actor,
2. the transformation identifies all paths to the asset,
3. and for every path, identifies the credentials that the actor is lacking;
4. for each missing credential, a new transformation is started recursively;
5. after obtaining all necessary credentials, the actor can reach the location of the goal asset, and perform an action to obtain it.

In the following subsections, we present for each of the model elements discussed in Section 3.2, how they are transformed into an attack representation. For each transformation we show the part of the system model that triggers the translation as well as the generated part of the attack model. For the system models we use the same graphical representation as shown in Section 3.6 and Figure 3.6. For attack models we use a special notation that represents parts of the attack as circles, and invocations of the transformation as rectangles.

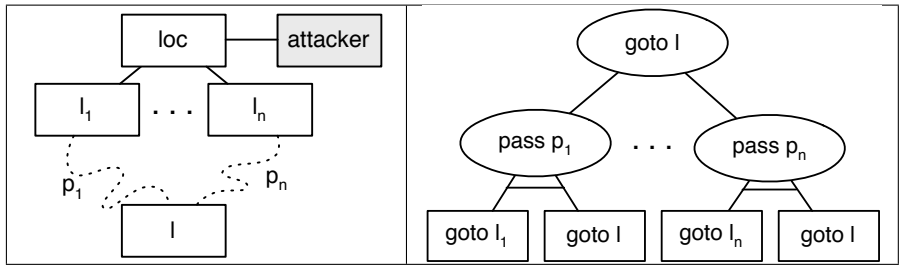


Figure 5.1: Reaching a location. For every path from an initial location *loc* to a target location l, we generate an attack that takes the first step and then continues to l.

5.1.1.1 Locations

A location is transformed into a disjunction of all possible paths from the locations already reached by the attacker to the location in question. Whenever traversing a path requires new credentials due to some policy, we recursively invoke the attack transformation, which ensures that the attacker obtains the necessary credentials to pass the path.

The transformation pattern is shown in Figure 5.1. For every possible path we first generate one step to the first node of the path, followed by a recursive invocation of the transformation for going to the target location.

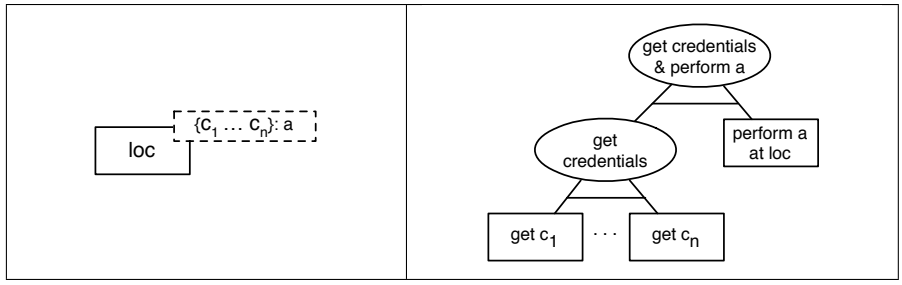


Figure 5.2: Transforming a policy. If the attacker lacks any credential to perform action a at the location *loc*, the transformation creates an attack that obtains that credential.

5.1.1.2 Policies

If the transformation at any point needs to create an action that is prohibited by a policy, for which the attacker lacks a credential, a new transformation is started to obtain this credential, resulting in a new attack representation. The transformation pattern is shown in Figure 5.2.

As mentioned above, many system models support predicates as credentials, for example, to express that the actor must possess a certain attribute. In the example shown in Figure 3.6, an actor must be trusted by Alice in order to be allowed to move to the location *Door*. Often, such a predicate is not a credential that can be obtained, as for trust. In this case, the transformation generates a social engineering action to "obtain" the predicate in question.

The variables in policies can be factored out before performing this transformation by identifying all sets of assets that fulfil a policy. For the example shown in Figure 3.6 and the location ATM, the possible sets of assets are the card and the pin at Alice or at Charlie.

We assume that the transformation generates all necessary steps for obtaining the necessary assets before performing the action. In the resulting attack representation, the root node of the attack representation for obtaining the necessary credentials will be to the left of the root node for performing the following actions, expressing an ordering as described above.

5.1.1.3 Data

Data represents intangible assets, such as passwords or pins. For obtaining data, a conjunction is generated where the first goal for the attacker is to reach the location of the data. There, an action in the attack representation will be generated that depends on where the data is contained:

- If the data is contained in a location, then an input will be generated; or
- If the data is contained at an actor, then a social engineering action will be generated.

If the goal data is contained in an item i , the transformation generates the conjunction of several actions:

- Obtain the item and then obtain the data from the item; or

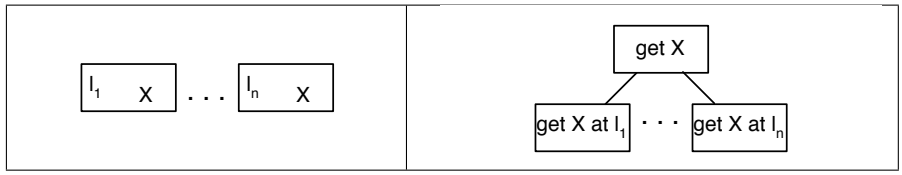


Figure 5.3: Items and data may be available from different locations. For each of these locations, the transformation generates a separate attack path to obtain the asset. The transformation will generate attacks to obtain all necessary credentials, and then input the asset.

Figure 5.4: To obtain an asset from a location, the transformation generates the necessary attack to go to the asset's location, then obtains any required credentials, and finally inputs the asset.

- Obtain the data from the item directly.

The difference between the two options is that the first option represents the case that the attacker obtains the containing item itself and then obtains the data, while the second option represents the case that the attacker removes the data or item in place.

For the example of the workstation mentioned at the end of Section 3.2 this would mean that the attacker either steals the hard drive containing the file, or that he extracts the file from the hard drive.

5.1.1.4 Items

Items represent tangible assets, such as the aforementioned workstation, hard-disk, or an access card. Just as for data, we generate a conjunction that first contains a node that represents the attacker reaching the location of an item. Then, an action in the attack representation will be generated that depends on the kind of location that contains the item:

- If the item is contained in a location, then an input will be generated;

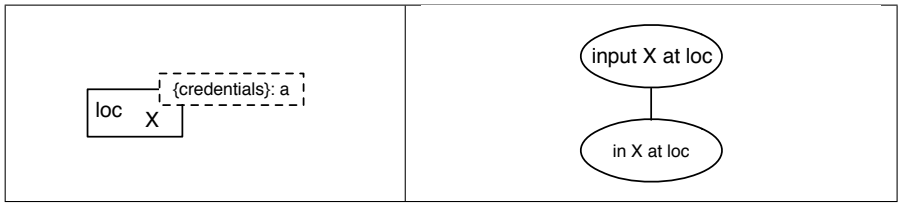


Figure 5.5: To obtain an asset that is directly contained at a location, the transformation simply generates an input. Note that the necessary credentials have been obtained before invoking this transformation (Figure 5.4).

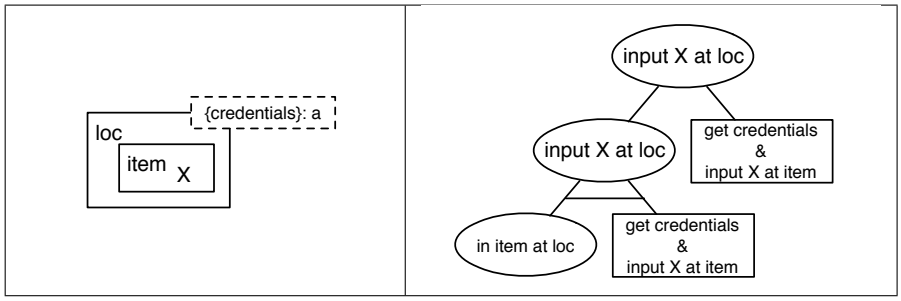


Figure 5.6: To obtain an asset that is transitively contained at a location, the transformation first obtains the item containing the asset and then recursively invokes the transformation.

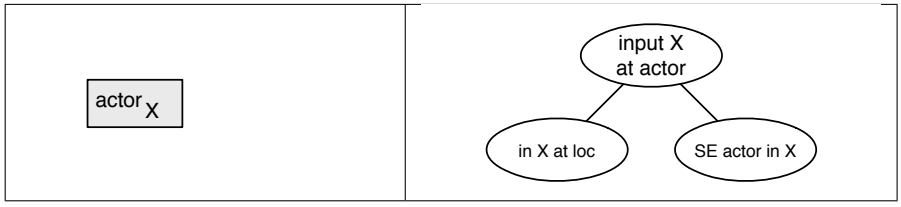


Figure 5.7: Obtaining an asset from an actor is almost the same as for locations; the only difference is that assets can be obtained by social engineering. The transformation generates a special social engineering action, which is not further defined. Refining this action depends on the context of the action such as, *e.g.*, the involved actors; this is left to later phases that consume the generated attack.

- If the item is contained at an actor, then a disjunction of a social engineering action or an in action will be generated, where the latter represents

an attempt of stealing the item.

If the goal item is contained in another item, the transformation generates the conjunction of several actions:

- Obtain the item and then obtain the goal item from the item; or
- Obtain the goal item from the item directly.

The difference between the two options in the generated disjunctions is that the first option represents the case that the attacker obtains the containing item itself, while the second option represents the case that the attacker removes the data or item in place.

For the example of the workstation mentioned before this would mean that the attacker either steals the workstation containing the hard-drive, or that he extracts the hard-drive from the workstation.

5.1.1.5 Triggering the Transformation

In general the transformation will be triggered by a certain asset being off-limits for an attacker. The transformation iterates over a specified set of actors available in the system model, and generates for each of these actors all possible attacks for how they can obtain the asset. The triggering transformation for an asset X is get X. While transforming the system model into an attack model, the transformation keeps track of the attacker, the location reached, and the assets obtained. The attacker may already possess assets before starting the transformation; this is specified in the system model.

5.1.2 Adding Data Mobility

So far we have assumed assets to be at static locations. This assumption simplifies both the transformations for attack generation and the structure of the generated attacks; instead of having to consider all the locations that an asset can reach by means of actors or processes, we only have to consider the locations where data is available in the model. We now discuss how to loosen this restriction.

In Subsection 5.1.1.3 and Subsection 5.1.1.4 the transformations described assume that the data is available from a number of locations in the model, either directly or transitively. The main transformation starting the generation of sub-attacks is shown in Figure 5.3. When adding data mobility, we are interested in which *other* locations the assets are able to reach, either by means of processes (for virtual assets) or by means of actors (for real-world assets).

The transformation for data mobility works in reverse compared to the transformations we have presented in the previous section. Before being able to generate an attack, we need to perform three steps:

1. Identify who is able to move the asset;
2. Identify how to trigger the movement; and
3. Identify which locations the asset can reach.

The result of these steps is an attack that triggers the movement, and a set of locations that the asset can reach; these locations can then be used as input to the transformation shown in Figure 5.3.

The main task lies in identifying who can trigger the movement and how. Beyond these steps, adding data mobility does not add to the transformation, but to the complexity of the generated attack model.

5.2 Policy Invalidation

After presenting the overview of the graphical transformation in the previous section, in this section we present the formalism behind the generation of attack trees by invalidating policies. Attack trees are chosen as a succinct way of representing attacks; they are defined by

Definition 5.1 (Attack trees and operations.) An attack tree is a tree $AT := (N_i \dot\cup N_l, n, E, L)$, containing inner nodes $N_i := N_\wedge \dot\cup N_\vee$ and leaf nodes N_l, a root node $n \in N_i$, directed edges $E \subseteq N_i \times N_i \dot\cup N_l$, and a labelling function $L := N \to \Sigma^\star$. A node in an attack tree is a conjunction ($N_\wedge$) or disjunction ($N_\vee$) of sub-attacks, or a basic action in an attack (N_l). Let $\mathcal{N}^{label}$ be the attack tree that only contains one node n that is mapped by L to *label*. For $AT_1 = (N_1, n_1, E_1, L_1)$ and $AT_2 = (N_2, n_2, E_2, L_2)$, $kind \in \{\vee, \wedge\}$, *label* being a string, and $n \in N_{kind}$, we define the addition of attack trees as

$$AT_1 \oplus^{label}_{kind} AT_2 := (N_1 \cup N_2 \cup \{n\}, E_1 \cup E_2 \cup \{(n, n_1), (n, n_2)\}, n, L_1 \cup L_2 \cup \{(n, label\})$$

Note that we assume an implicit, left to right order for the children of conjunctive nodes. For example, an attacker first needs to move to a location before being able to perform an action.

Policy invalidation operates on global, organisational policies. Like local policies, a global policy consists of required credentials and enabled actions. On a high level, our approach for invalidating a policy consists of four basic steps:

1. Choose the policy to invalidate, and identify the possible actors who could do so; these are the potential attackers.
2. Identify a set of locations where the prohibited actions can be performed. Since there might be several possible actions, this results in a set of pairs of location and action.
3. Recursively generate attacks for performing these actions. This will also identify required assets to perform any of these actions, and obtain them.
4. Finally, move to the location identified in the second step and perform the action.

It should be noted that all rules specified below either block if no valid result can be computed, or return an empty attack tree, for example, if no credentials are required. The rules take as input an infrastructure component $\mathcal{I}$, which has been defined in Section 4.2, and an actor component $\mathcal{A}$, which stores identities, locations, and assets collected and reached by an actor during an attack. Also note that we extend rules from working on singular elements to sets by unifying the results of rule applications.

5.2.1 Identify Attackers

The overall rule for starting the attack generation from a global policy is shown in Figure 5.8. After having computed the unification of the global policy and the set of all actors, we identify the set of attackers by means of function *getAttacker*, which replaces a variable with the identified bindings, or returns an explicitly specified attacker:

$$getAttacker_{\mathcal{I}}(a, \sigma) \quad := \quad \begin{cases} \{a\} & \text{if } a \in \mathsf{N}_a \\ \sigma(a) & \text{if } a \in Vars \end{cases}$$

$$\frac{\begin{array}{c}\sigma = unify_{\mathcal{I}}(Actors, credentials) \\ attackers = getAttacker_{\mathcal{I}}(actor, \sigma) \qquad goals = applicableAt_{\mathcal{I}}(credentials, enabled, \sigma) \\ \mathcal{I}, attackers, goals \vdash_{goal} trees \qquad \mathcal{T} := \oplus_{\vee}^{\text{“perform any actions”}} trees\end{array}}{\mathcal{I}, not(actor, credentials, enabled) \vdash_{P} \mathcal{T}}$$

Figure 5.8: Attack generation starts from the global action-based policy $not(actor, credentials, enabled)$. Attack trees are generated for all possible policy violations. As every attack tree represents a violation of the policy, the resulting attack trees are combined by an *or* node.

$$\frac{\mathcal{I}, \mathcal{A}, goto(location) \wedge perform(action) \vdash_{GP} \mathcal{T}}{\mathcal{I}, \mathcal{A}, (location, action) \vdash_{goal} \mathcal{T}}$$

$$\frac{\mathcal{I}, \mathcal{A}, goto(l) \vdash_{goto} \mathcal{T}_{goto}, \mathcal{A}' \qquad \mathcal{I}, \mathcal{A}', perform(a) \vdash_{perform} \mathcal{T}_{action}, \mathcal{A}''}{\mathcal{I}, \mathcal{A}, goto(l) \wedge perform(a) \vdash_{GP} \mathcal{T}_{goto} \oplus_{\wedge}^{\text{“goto } l \text{ and perform } a\text{”}} \mathcal{T}_{action}, \mathcal{A}''}$$

Figure 5.9: For each identified goal (consisting of a location and an action) an attacker moves to the location and performs the action. The rules result in an attack tree and a new state of the attacker, which includes the obtained keys and reached locations.

5.2.2 Identify Target Locations

Assuming there is an attacker, we then compute all locations at which one of the actions in *enabled* could be applied using the credentials specified in the policy. The function *applicableAt* identifies all locations in the system model at which one of the actions is applicable and returns goals as pairs of actions and locations.

5.2.3 Attack Generation

Using the information about all possible attackers and the locations at which the actions are applicable, we proceed to the actual generation of attack trees. The rules in Figure 5.9 connect the identified goals with the generation of attack trees. For each goal, two attack trees are generated: one for moving to the location and one for performing the action. While moving to the location, new credentials may be required; as a result, the actor acquires new knowledge, which is stored in the actor component $\mathcal{A}$, which contains the obtained identities and

$$\frac{paths = getAllPaths_{\mathcal{I}}(\mathcal{A}, l) \qquad \mathcal{I}, \mathcal{A}, paths \vdash_{path} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\vee}^{\text{“find path to } l\text{”}} trees}{\mathcal{I}, \mathcal{A}, goto(l) \vdash_{goto} \mathcal{T}, \mathcal{A}'}$$

$$\frac{missing = missingCredentials_{\mathcal{I}}(\mathcal{A}, path) \qquad \mathcal{I}, \mathcal{A}, missing \vdash_{credential} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\wedge}^{\text{“get credentials”}} trees}{\mathcal{I}, \mathcal{A}, path \vdash_{path} \mathcal{T} \oplus_{\wedge}^{\text{“get credentials and pass path”}} \mathcal{N}^{\text{pass path}}, \mathcal{A}'}$$

Figure 5.10: Going to a location and performing an action is broken down into two attack trees. The function *getAllPaths* returns all paths from the current locations of the actor to the goal location l, and the resulting attack trees are combined as alternatives for reaching this location. The resulting actor has obtained the necessary credentials and reached the location l.

credentials, and reached locations.

The rules in Figure 5.10 and Figure 5.11 simulate actions of attackers in the modelled organisation: moving around, performing actions, and obtaining credentials. In doing so, attack trees are created for every single action of the attacker, and the resulting trees are combined in the overall attack tree.

The function *missingCredentials* uses the unification described above to match policies with the assets available in the model. This implies that all assets that can fulfil a policy are identified; the attack generation then generates one attack for each of these assets, and combines them with a disjunctive node.

Another global location-based policy could forbid employees to obtain money that has been “owned” by a customer. In this case, the processes defined in Section 3.6 for the set-top box and the bank computer would become important, as they allow to transfer money from Alice’s to Charlie’s account. When invalidating this global policy, the analysis computes the flow of assets both through the world and the network layer.

5.2.4 Post-Processing Attack Trees

The generated attack trees also often contain duplicated sub-trees, due to similar scenarios being encountered in several locations, for example, the social engineering of the same actor, or the requirement for the same credentials. This

$$\frac{i \notin identities \implies \mathcal{T} = \mathcal{N}^{\text{obtain identity } i}}{\mathcal{I}, (identities, locations, assets), identity\ i \vdash_{credential} \mathcal{T}, (identities \cup \{i\}, locations, assets)}$$

$$\frac{\begin{array}{c}\mathcal{A} = (identities, locations, assets) \wedge a \notin assets \implies \\ goals = availableAt_{\mathcal{I}}(a) \qquad \mathcal{I}, \mathcal{A}, goals \vdash_{goal} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\vee}^{\text{“get a”}} trees\end{array}}{\mathcal{I}, \mathcal{A}, asset\ a \vdash_{credential} \mathcal{T}, \mathcal{A}'}$$

$$\frac{\mathcal{I}, \mathcal{A}, predicate\ p(arguments) \vdash_{predicate} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\vee}^{\text{“fulfil predicate } p\text{”}} trees}{\mathcal{I}, \mathcal{A}, predicate\ p(arguments) \vdash_{credential} \mathcal{T}, \mathcal{A}'}$$

Figure 5.11: Depending on the missing credential, different attacks are generated. If the actor lacks an identity, an attack node representing an abstract social engineering attack is generated, for example, social engineering or impersonating. If the missing credential is an asset, the function *availableAt* returns a set of pairs of locations from which this asset is available, and the according **in** actions. If the missing credential is a predicate, a combination of credentials fulfilling the predicate must be obtained.

is not an inherent limitation, but often makes the attack trees unnecessarily big and cluttered. Similar to [VNN14], a post-processing of attack trees can simplify the result.

5.3 Analysing the Running Example

In this section we demonstrate the mechanism of the attack generation by relating it to the running example. The attack tree shown in Figure 5.12 is generated from the example scenario.

As mentioned in the previous section, we define as the global policy in our running example that an employee is not allowed to use a customer's card to obtain money:

$$not(\{X\}, \{isEmployee(X), card[(owner, Y)], isCustomer(Y)\}, \{\mathbf{in}\})$$

Using the rule from Figure 5.8, we compute a substitution for the variables X and Y to be

$$\sigma = [X \mapsto Charlie, Y \mapsto Alice]$$

The substitution σ maps X to Charlie because he has the role employee, and Y to Alice as she has the role customer. In the next step, the attacker is identified to be the mapping of X in σ, namely Charlie, and then we identify the possible locations where the global policy can be violated. Based on the system specification from Figure 3.6, the only location with a policy restricting the **in** action is the money location at the ATM A1. There is therefore only one possible location and action pair, namely $\{(A1, \mathbf{in})\}$, and the next step is to invoke $\vdash_P$ from Figure 5.10 to generate the attack tree for moving to $A1$ and performing the **in** action.

Going to the location does not require additional credentials, but performing the **in** action does. The *missingCredentials* function returns the card and the pin, which combined with the requirement from the goal policy, that the owner of the card must be Alice, implies that the needed credentials are Alice's card and her pin. For each of these credentials the second rule $\vdash_{credential}$ in Figure 5.11 identifies where they are: Alice has the card and the pin, and the pin code is also stored in the card.

As a result, our approach generates an attack tree for going to the location "home", and in doing so the attacker must fulfil the policy "trustedBy(Alice)", meaning that he must impersonate somebody trusted by Alice. Then the attacker can either "input" the card and the pin, or only the card and then try to extract the pin code from the card.

The stealing and the extraction of the pin code are not represented in the model since they are context and technology dependent. In a given scenario, they can be instantiated with the matching "real" actions. After the assets have been obtained, the attacker moves to the ATM location and performs the action.

For the first possible attack, Charlie would use his own card and pin; this does not require further credentials. For the second possible attack, Charlie needs to obtain the pin and the card from Alice. Alice's location is *Home*, and to pass the path to this location, Charlie must fulfil the predicate *trustedby(Alice)*. This results in an action *social engineer Alice move Door*, which could in a later phase, for example, be translated into a forceful entrance or pretending to be somebody who Alice trusts or is likely to let in her home. Once the location *Home* has been reached, Charlie has several options for obtaining the card and the pin:

- Social Engineer Alice to give him the card and the pin;

- Input card from Alice (stealing); and
- Input the pin from the card (skimming).

The generated attack represents all combinations hereof; some parts of the tree can be pruned or simplified in a later phase similar to [VNN14]. Once the card and the pin have been obtained, Charlie moves to the location ATM and inputs the asset *cash*.

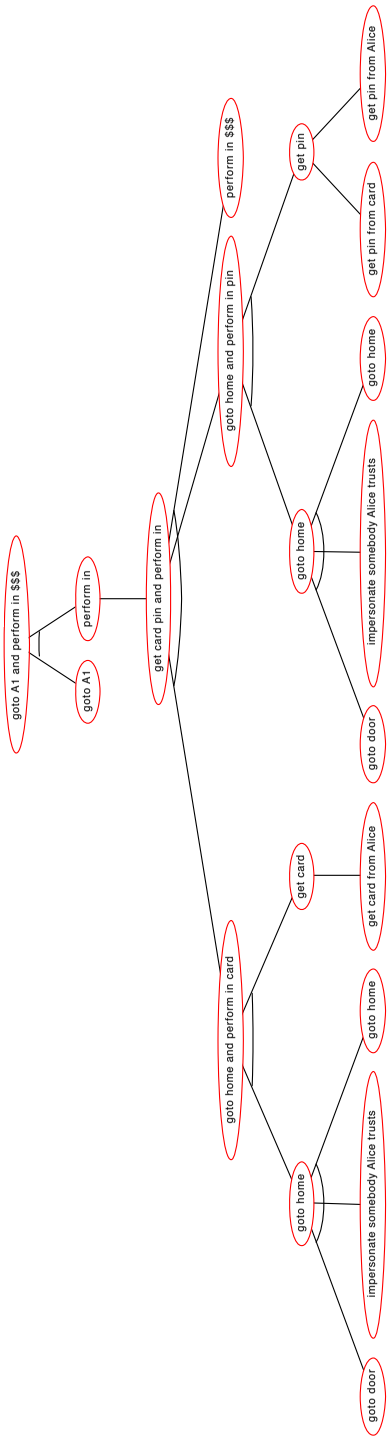


Figure 5.12: Attack tree generated by the prototype implementation for the example shown in Figure 3.6.

The resulting attack tree is shown in Figure 5.12. Not surprisingly, the transformation result contains identical sub-trees due to identical assets to obtain or identical patterns being transformed. Similar to the actions for obtaining items, these could be simplified by a follow-up pass.

5.4 Implementation

The implemented tool takes as an input a scenario describing a system model of an organisation and a policy to invalidate. The policy, as described in Section 3.3.2, can be either location-based or action-based.

The input has an XML format. The XML structure along with a description of the different elements and attributes can be found in Appendix A. The tool has been implemented in Scala, but has not been released publicly yet. It generates an attack tree, based on the input, and represents it textually in XML format. The ADtool [KS12a; KS12b] is then used to turn the XML into a graphical representation of the generated attack tree.

5.5 Concluding Remarks

In this chapter we have shown an approach to attack tree generation *including* human factors from a system model. While the notion of "human factor" could be stronger, our approaches are flexible enough to support all kinds of human factors that can be instantiated once an attack has been identified.

In the work described in this chapter, we only consider the pure transformation of system models to attack trees. An essential next step in risk assessment is to evaluate the risk and impact of the different attack vectors, for example, by annotating the attack tree with metrics and performing analyses on them [AN15]. This mapping can be achieved by associating the elements' identifiers with relevant metrics. These metrics can represent any quantitative knowledge about components, for example, likelihood, time, price, impact, or probability distributions. The latter could describe behaviour of actors or timing distributions. For the transformation described in this chapter these metrics are irrelevant, but they can be evaluated on the generated attack trees.

5.5.1 Complexity, Soundness and Completeness

Important issues in the model as well as in the attack generation are complexity as well as soundness and completeness of the attack generation with respect to the model. All the three properties are directly related to the usability of the framework. Soundness and completeness ensure that the results are reliable, and complexity impacts scalability of the approach; scalability implies that the results can be computed also for real-world scenarios.

In the attack generation described in Section 5.2.3 we assume that an attacker has a set of keys and a set of identities. Throughout the generation process, the attack generation keeps track of the locations reached by an attacker. Using these properties, attackers can obtain new identities, new keys, and reach new locations, and doing so will give them the possibility of acquiring even more knowledge or access. Since the attack generation is a white-box analysis, where all the knowledge about the analysed model is available through an oracle, the overall complexity of attack generation is in theory cubic in the size of the model; in practice, the number of identities and keys can be expected to be much smaller than the size of the model, reducing the complexity significantly.

The attack generation is sound with respect to the model, that is all generated attack trees establish a sequence of actions that will lead to a validation of a goal in the modelled organisation. Like counter-example traces in model checking, this means that the result of the attack generation shows one or several ways in which the goal policy can be broken; several ways are represented as alternatives in the generated attack tree.

The attack generation is also complete with respect to the model, that is all attacks that are possible in the model will be found. The attack generation starts from the goal policy, which states what is considered as an attack on the modelled organisation; the generation then identifies all possible actions of breaking the goal policy. For each of these actions, all possible attacks are identified for performing them, and so on, until the necessary assets are acquired.

The important point in this discussion is “with respect to the model”. Only attacks in the model can be identified, and if a necessary asset, action, or location is not contained in the model, it cannot and will not be considered in any generated attack.

5.5.2 Precision

The combination of system model and automated generation enables us to trade in precision of the model for details in the attack trees. For example, the modelling of the ATM is very imprecise in the example from Figure 3.3. In that example the ATM has a policy that allows inputting the money, once the correct pin is provided. Therefore, attacks derived from this way of modelling of the ATM would need a card and the corresponding pin. The connection between the user and the obtained money is direct. Other attacks, for example shoulder surfing, would not be generated by analysing the model.

A more detailed description would be:

$$\begin{aligned} P_U \quad := \quad & \mathbf{out}\,((\text{“}card\ \&\ pin\text{”}, card, pin, user))\,@ATM. \\ & \mathbf{in}\,((\text{“}card\text{”}, card))\,@U. \\ & \mathbf{in}\,((\text{“}amount\text{”}, amount))\,@U \end{aligned}$$

$$\begin{aligned} P_{ATM} \quad := \quad & \mathbf{in}\,((\text{“}card\ \&\ pin\text{”}, !card, !pin, !user))\,@ATM. \\ & \mathbf{read}\,((\text{“}pin\text{”}, pin))\,@card. \\ & \mathbf{in}\,((\text{“}amount\text{”}, !amount))\,@ATM. \\ & \mathbf{out}\,((\text{“}card\text{”}, card))\,@U. \\ & \mathbf{out}\,((\text{“}amount\text{”}, amount))\,@U \end{aligned}$$

This process definition represents that an actor puts the card and the pin code into the ATM and receives back money after a check with the bank. In this model, the attack tree generator is able to find out that one can obtain the pin code from the ATM, since it is input into the system.

A third, even more precise, approach would be having a connection from the user, through the keyboard, to the ATM, and respectively, connection between the user and the ATM when the money is output:

A process at the card reader (P_{CR}) can be defined as follows:

$$\begin{aligned} P_{CR} \quad := \quad & \mathbf{in}\,((\text{“}card\text{”}, !card))\,@CR. \\ & \mathbf{read}\,((\text{“}pin\text{”}, !pin))\,@card. \\ & \mathbf{out}\,((\text{“}pin\text{”}, pin))\,@CR \end{aligned}$$

Accordingly, the process running at the ATM machine (P_{KB}) would look like this:

$$
\begin{aligned}
P_{ATM} \quad := \quad & \mathbf{in}\,((\text{“}pin\text{”}, !pin))\,@KB. \\
& \mathbf{in}\,((\text{“}pin\text{”}, pin))\,@CR. \\
& \mathbf{in}\,((\text{“}amount\text{”}, !amount))\,@KB. \\
& \mathbf{out}\,((\text{“}amount\text{”}, amount))\,@Money. \\
& \mathbf{out}\,((\text{“}card\text{”}, card))\,@CR
\end{aligned}
$$

$$
\begin{aligned}
P_{U} \quad := \quad & \mathbf{out}\,((\text{“}pin\text{”}, pin))\,@KB. \\
& \mathbf{in}\,((\text{“}amount\text{”}, amount))\,@KB. \\
& \mathbf{in}\,((\text{“}amount\text{”}, !amount))\,@Money
\end{aligned}
$$

Yet another level of precision would be to define a process on the payment card that checks the pin code. However, the more sophisticated the model is, the more complex the attack generation and respectively the size of the attack tree would be. In order to not excessively complicate the further attack tree analyses, it is important for the modeller to model the right level of details, depending of the nature of the attacks that need to be identified.

Chapter 6

Evaluation

In this chapter we illustrate how our approach is applied to risk assessment of cloud infrastructures. While the assessment is similar to the one shown before for the IPTV case study, the system is more complex. A typical cloud infrastructure potentially contains thousands of nodes which are highly interconnected and dynamic.

After presenting risk assessment for a simplified scenario from the cloud case study, we conclude the chapter with a discussion of our approach considering the related work presented in Chapter 2.

6.1 Cloud Computing

We start with an overview of the different components in a typical cloud infrastructure. After that, we present a simplified scenario from the cloud case study and discuss how we model the different components, summarising them for better visibility as an informal graphical representation of the model. As a cloud environment is a very dynamic system, quite some attention is devoted to the modelling of processes, which is the main force of achieving the dynamic behaviour. Finally, we present the attack generation from the model.

As mentioned earlier, cloud infrastructures are typically complex and dynamic. The success of cloud infrastructure is exactly in this dynamic and flexible structure. At the same time, however, this flexibility of the interconnections between physical and virtual machines makes risk assessment a hard task. There are studies highlighting the risks and threats in such environments [ENI09; Clo10]. While there is quite a history of network models in security, these models have generally been limited to the technical parts of an organisation's infrastructure, typically representing computer networks and hops of an attacker from one node to another.

Together with the human factor the cloud infrastructures become even more sophisticated socio-technical systems and thus their risk assessment even more complicated. Using our approach, one can model the typical components of a cloud environment, for example, switches, routers, physical servers, virtual machines, infrastructure details like buildings, rooms, doors, etc [Nid+15]. The model includes also the connectivity relations among those components as well as any access control.

Below we set the scene by shortly presenting a typical structure of a cloud environment. For a better overview, we describe the cloud infrastructure in three layers:

- *Technical layer*, which includes three sublayers: hardware (servers, storage, internal and external network connections), virtualisation (hypervisors, virtual machines, network and storage components together with their configurations) and software (operating systems, middleware, applications and services).
- *Physical layer* is of course closely related to the technical layer. It represents the physical connections of the elements mentioned above. Respectively, it consists of buildings, rooms and the respective access control.
- *Social layer* considers the social and human factor.

6.1.1 Scenario

In this section we apply our modelling approach to the scenario presented in Figure 6.1. The physical infrastructure consists of two rooms and a hallway. Doors connect the outside world with the hallway, the hallway with the internal room and the internal room with the data centre, respectively. Both rooms have a window.

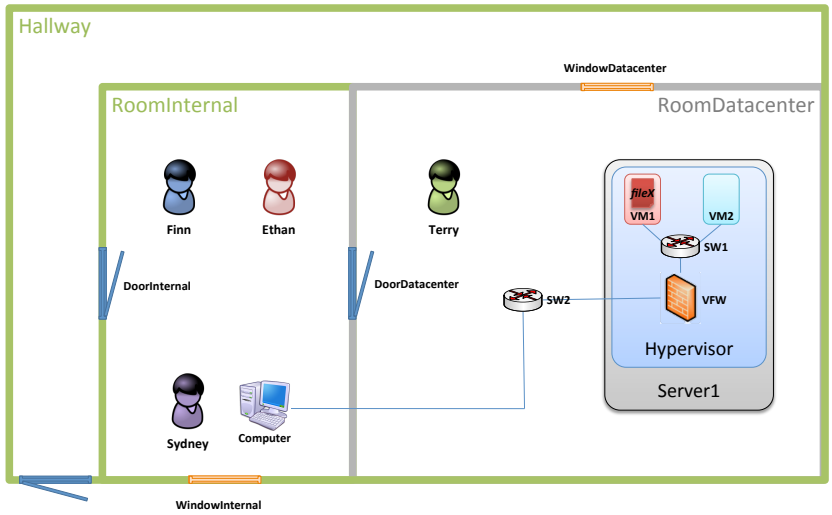


Figure 6.1: Highly simplified scenario of a private cloud environment with various actors.

In the data centre there is a physical server on which there are two virtual machines running (*VM1* and *VM2*) together with a switch *SW1* and a firewall *VFW*, all on top of a hypervisor.

Our actors include Sydney, Terry, Finn, and Ethan. Sydney is a system administrator in the IT support, who not surprisingly has full logical access to the whole cloud infrastructure. Terry is a technician in the IT support, and thus has a physical access to the data centre. Finn works as a part of the finance department, which uses *VM2* for their work. Finally, Ethan is a member of the organised crime department.

6.1.2 Modelling

We relate the elements of the cloud environment to the components in our model as described below. The hardware elements like physical servers and routers

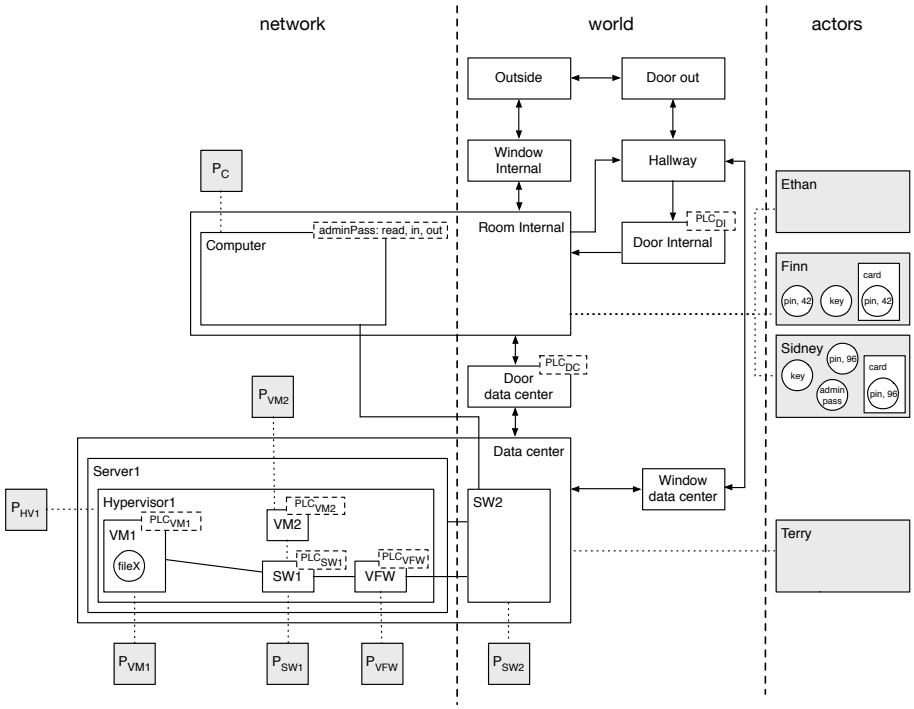


Figure 6.2: Abstract representation of the system model of the cloud scenario. The directed edges correspond to physical connections, and the undirected ones correspond to logical connections. The white rectangles represent locations, while the grey rectangles represent actors. The circles represent data. The actors have different items or data in their tuple space. The dashed rectangles at the upper right corner of some of the nodes represent policy labels (see below for the corresponding policy definitions). The grey squares connected with dotted lines are the processes at each network location.

are modelled as items, each of which has a location. Items, as for example, the computer in the internal room is an item stored in the tuple space of the internal room. Similarly, the virtual components are modelled also as locations. For example, each of the virtual machines *VM1* and *VM2* is modelled as a location in the tuple space of the physical server *SW1* it is running on. Here in addition, as we will see later, processes come into the picture and processes are associated/executed on certain locations.

Elements from the physical infrastructure are naturally modelled as locations.

The access control is expressed using policies.

The actors in the model are modelled as they appear in the scenario: initially Finn, Ethan and Sydney are located in the internal room, while Terry is in the data centre.

Figure 6.2 illustrates the system model of the scenario described above in Section 6.1.1. All actors have roles assigned to them, which are not shown in the model, but can be used in policies to control access for actors belonging to a specific group, *i.e.*, having a specific role assigned.

Each network node has policies and processes assigned. The policies are used to represent who can communicate to that node and what form the messages should be in. The processes represent handling of IP traffic, and define how the node should react when triggered by a specific message format.

Policy Modelling. In our example from Figure 6.2, the $SW1$ node represents a switch that can communicate with the nodes $VM1$, $VM2$, and VFW.

The respective policies are defined as follows:

$$\begin{aligned} PLC_{SW1} &:= \{VM1 \,|\, VM2 \,|\, VM3 \,|\, VFW : \mathbf{out}\} \\ PLC_{VM1} &:= \{SW1 : \mathbf{out}(\text{“}IP\text{”}, , , , ,)\} \\ PLC_{VM2} &:= \{SW1 : \mathbf{out}(\text{“}IP\text{”}, , , , ,)\} \\ PLC_{VFW} &:= \{SW1 \,|\, SW2 : \mathbf{out}\} \\ PLC_{DI} &:= \{key : \mathbf{move}\} \\ PLC_{DC} &:= \{card[(pin, X)], pin(X) : \mathbf{move}\} \end{aligned}$$

PLC_{SW1} allows the three virtual machines and the virtual firewall VFW on Server1 to output at the switch $SW1$. For readability reasons we use the symbol | which works as a disjunct; it is not part of the policy specification language; in principle we should define one policy per disjunct. PLC_{VM1} and PLC_{VM2} express that the switch $SW1$ is allowed to output at each of the respective virtual machines. In other words, the switch communicates internally with each of the virtual machines.

On the physical layer, the access through the door to the internal room is restricted by a key lock, while the access to the data centre is restricted by a card and the corresponding pin code for that card.

Process Modelling. As said above, the cloud case study represents a rather dynamic environment, whose flexibility needs to be modelled. We model the

dynamics of the cloud environment by defining processes. For the processes in this section we allow for general tuples, as in standard Klaim [DFP98b], to model network traffic.

The processes in the cloud case study can be classified in two main categories: processes that represent handling of IP traffic, and processes that represent hypervisor functionality, *e.g.*, the migration of virtual machines.

In a more simplistic world a process usually triggers certain actions. In our case, however, it is more complicated, resulting in a process triggering another process. Therefore, network traffic is modelled as a sequence of processes triggered on the various nodes along a route. A service is represented as the final process in the chain, *i.e.*, at the destination node, which triggers some computation and the produced return message.

In order to illustrate the process modelling let's assume that $VM1$ and $VM2$ have IP addresses 192.167.1.1 and 192.167.1.2, respectively. Both of them are part of the network 192.167.1.0/24, and, as Figure 6.2 shows, are connected by a switch SW1 with a default gateway on VFW.

A packet is represented by a tuple with 6 elements: a tag "IP", the source address and the source port, the destination address and port, and the message. Of course this is adjustable according to the scenario in question. The $\sim$ is an operator representing the matching of the addresses, which works similarly to pattern matching.

A process running at the VM1 (P_{VM1}) would look like this:

$$\begin{aligned}&\textbf{in}\,(\text{“}IP\text{”}, !srcAddr, !srcPort, !dstAddr, !dstPort, !msg)\,@VM1.\\&\textbf{out}\,(\text{“}IP\text{”}, srcAddr, srcPort, dstAddr, dstPort, msg)\,@SW1\end{aligned}$$

The processes at $VM2$ (P_{VM2}) is similar:

$$\begin{aligned}&\textbf{in}\,(\text{“}IP\text{”}, !srcAddr, !srcPort, !dstAddr, !dstPort, !msg)\,@VM2.\\&\textbf{out}\,(\text{“}IP\text{”}, srcAddr, srcPort, dstAddr, dstPort, msg)\,@SW1\end{aligned}$$

At the switch (P_{SW1}), there are three processes, which would then look like this:

$$\begin{aligned}&\textbf{in}\,(\text{“}IP\text{”}, !srcAddr, !srcPort, 192.167.1.1, !dstPort, !msg)\,@SW1.\\&\textbf{out}\,(\text{“}IP\text{”}, srcAddr, srcPort, 192.167.1.1, dstPort, msg)\,@VM1\ \ |\\&\textbf{in}\,(\text{“}IP\text{”}, !srcAddr, !srcPort, 192.167.1.2, !dstPort, !msg)\,@SW1.\\&\textbf{out}\,(\text{“}IP\text{”}, srcAddr, srcPort, 192.167.1.2, dstPort, msg)\,@VM2\ \ |\\&\textbf{in}\,(\text{“}IP\text{”}, !srcAddr, !srcPort, !dstAddr, !dstPort, !msg)\,@SW1.\\&\textbf{out}\,(\text{“}IP\text{”}, srcAddr, srcPort, dstAddr, dstPort, msg)\,@VFW\end{aligned}$$

The data routing uses the general IP routing method, where the destination network is matched and the traffic is replicated to the next hop. In case several paths are possible, the traffic will be sent through all available (acyclic) paths. Routing is modelled using the worst-case assumption; assuming any acyclic path might be taken. Processes are ordered by destination addresses, thus the first match is the route taken.

Even though hubs are not widely used nowadays, we would like to shortly discuss the difference between hubs and switches when it comes to defining the processes. Modelling hubs would allow more conservative risk assessment. On the one hand, a switch knows the destination address, *i.e.*, where to send the packet to. On the other hand, a hub passes all the traffic to all the directly connected devices without caring about the destination address. It is then the devices that check the destination address of the packets and accept only those which are meant for them.

With the above being said, a process at a hub would look like the following:

$$\begin{array}{l}\textbf{in}\,(\text{“}IP\text{”}, !srcAddr, !srcPort, !dstAddr \sim 192.167.1.0/24, !dstPort, !msg)\,@HUB.\\ \textbf{out}\,(\text{“}IP\text{”}, srcAddr, srcPort, dstAddr, dstPort, msg)\,@\{VM1, VM2\} \quad |\\ \textbf{in}\,(\text{“}IP\text{”}, !srcAddr, !srcPort, !dstAddr, !dstPort, !msg)\,@HUB.\\ \textbf{out}\,(\text{“}IP\text{”}, srcAddr, srcPort, dstAddr, dstPort, msg)\,@VFW\end{array}$$

The notion $out(t)@\{l_1, l_2, ..., l_n\}$ represents that the tuple is output to all the locations mentioned inbetween the curly brackets.

Firewalls play an essential role in network security. Normally a firewall separates not only inside from outside, but also defines sections within a network. Similarly to walls in a building, their main function is to isolate a particular section, thus limiting the scope of a potential damage.

Recently, a firewall implements much more functionality than simple allow/deny access rules based on a protocol, an address, and a port. Nowadays, a firewall goes beyond its classical application, providing various services like NAT, load balancing, and even routing. Deep-packet inspection, protocol validation, external black-lists, and HTTP header filtering are also useful services of modern firewalls.

While all the above services affect network security, for simplicity reasons, in this chapter we focus on the traditional firewall functionality, which allows or denies access given a protocol, a source and destination address, and a port. Adding the additional features of a firewall would require more granular modelling using processes similar to the ones regarding the routing described above.

We present two ways of modelling a firewall. In the first one, the firewall is modelled as a natural extension of a router. Before showing the process defining the firewall, for a better comparison, we present the process definition describing a switch.

in ("IP", !$srcAddr$, !$srcPort$, !$dstAddr \sim 10.0.0.0/24$, !$dstPort$, !msg) @$SW1$.
out ("IP", $srcAddr$, $srcPort$, $dstAddr$, $dstPort$, msg) @$VM1$ |

in ("IP", !$srcAddr$, !$srcPort$, !$dstAddr \sim 10.0.1.0/24$, !$dstPort$, !msg) @$SW1$.
out ("IP", $srcAddr$, $srcPort$, $dstAddr$, $dstPort$, msg) @$VM2$

In the above, all traffic received for either of the attached networks will be repeated to the destination network. Now with a slight modification of the process definition above, we can express the firewall functionality by, for example, blocking the incoming traffic to $10.0.1.0/24$:

in ("IP", !$srcAddr$, !$srcPort$, !$dstAddr \sim 10.0.0.0/24$, !$dstPort$, !msg) @$SW1$.
out ("IP", $srcAddr$, $srcPort$, $dstAddr$, $dstPort$, msg) @$VM1$ |

in ("IP", 192.167.1.2, 80, !$dstAddr \sim 10.0.0.0/24$, !$dstPort$, !msg) @$SW1$.
out ("IP", 192.167.1.2, 80, $dstAddr$, $dstPort$, msg) @$VM2$ |

in ("IP", !$srcAddr$, !$srcPort$, !$dstAddr \sim 10.0.1.0/24$, !$dstPort$, !msg) @$SW1$.
out ("$DENIED$", $srcAddr$, $srcPort$, $dstAddr$, $dstPort$, msg) @$SW1$

The second way to model a firewall would look like this:

in ("IP", !$srcAddr \sim 10.0.1.0/24$, !$srcPort$, !$dstAddr$, !$dstPort$, !msg) @$VFW1$.
out ("$SENT$", $srcAddr$, $srcPort$, $dstAddr$, $dstPort$) @$VFW1$.
out ("IP", $srcAddr$, $srcPort$, $dstAddr$, $dstPort$, msg) @$SW2$ |

in ("IP", !$srcAddr$, !$srcPort$, !$dstAddr \sim 10.0.1.0/24$, !$dstPort$, !msg) @$VFW1$.
read ("$SENT$", $dstAddr$, $dstPort$, $srcAddr$, $srcPort$) @$VFW1$.
out ("IP", $srcAddr$, $srcPort$, $dstAddr$, $dstPort$, msg) @$SW1$

The first process above records the outgoing traffic. The constant "SENT" is output at $VFW1$ together with the destination address and port so the firewall keeps track of which destinations packages have been sent to. The second process represents the incoming traffic. The firewall checks whether packages have been previously sent to the received destination address and port. This allows to model the functionality of the firewall, which accepts replies only to sent messages.

As for the migration of virtual machines, it is exactly the hypervisor, which is responsible for it. Below is a process definition running on Hypervisor1 (P_{HV1}):

in ("IP", !$srcAddr$, !$srcPort$, $HV1$, $HV1Port$, ("$migrate$", !$vmname$, !$targethv$)) @$HV1$.
in (($vmname$, !$vmloc$)) @$HV1$
out ("IP", $srcAddr$, $targethv$, ("$migrated$", $vmname$, $vmloc$)) @$HV1$ |
in ("IP", !$srcAddr$, !$srcPort$, !$target$, !$targetPort$, !msg) @FVW.
out ("IP", "$HV1$", $target$, msg) @$SW2$

When the hypervisor receives a migrate command with a virtual machine name and a target server, it inputs the location of that virtual machine, serialises the virtual machine and sends a message to the target hypervisor.

Finally, at the target hypervisor, when a message for a migrated virtual machine is received, together with its name and location, the target hypervisor outputs the virtual machine at the specified location:

in (“IP”, !$srcAddr$, !$srcPort$, $HV1$, $HV1Port$, (“$migrated$”, !$vmname$, !$vmloc$)) @$HV1$.
out (($vnname$, $vmloc$)) @$HV1$

6.1.3 Attack Generation

In cloud environments, there are various categories of attacks, among others: stealing of confidential data, corrupt business operations or financial gains.

In practice, as many of the devices are sophisticated and flexible, nearly each of them could be an entry point of an attack, once an actor gains access to their console and has knowledge about the administrative password.

For simplicity, in our scenario from Figure 6.1, we define as a goal gaining access to a confidential document *fileX*.

Using the policy specification language described in Section 3.3 the goal would be defined as:

$$not(\{fileX\}@\{outside\})$$

The XML file used as an input describing the scenario presented in Section 6.1.1 (see also Figure 6.1 and Figure 6.2) can be seen in Appendix B. Since the actors Sidney and Terry have similar access rights, the respective attack trees generated for each of them are very similar. Analogical is the case with the actors Finn and Ethan. Due to their size, we show in Figure 6.3 only the attack tree generated for Terry since it is the smallest one. We would like to note that the leaves of the attack tree (*i.e.*, the basic actions) are not including details since they come from an attack pattern library. The further refinement of the actions results in a more detailed attack tree, which is naturally larger. The refinement of the attack tree shown below appeared to be too large to be readable in the hard copy version of this dissertation, therefore it can only be seen in the on-line version (after a significant zoom-in).

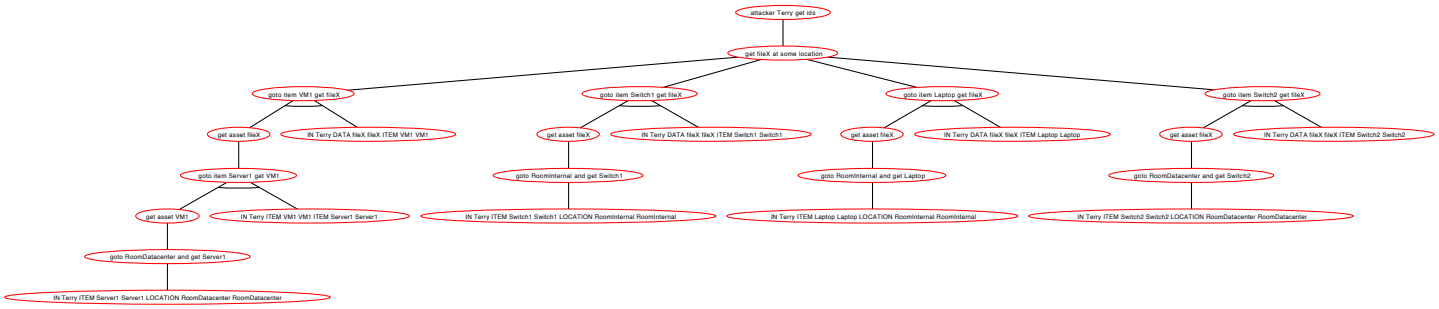


Figure 6.3: An attack tree generated for the actor Terry as a potential attacker.

6.2 Comparison with Related Work

After presenting the concepts of our approach, and applying it on the cloud case study scenario, here we relate our concepts to the related work presented in Chapter 2.

6.2.1 Representation

There are two main categories found to represent models: graphical and textual. The graphical representation can be divided in trees, graphs, diagrams, and maps, with the former two having clear mathematical properties that support formal analyses. Diagrams and maps help to communicate models to tool users. Plain textual representations are mainly used as an input for tools. An optimal model representation is probably a hybrid of these approaches. Our model is XML-based; this textual representation can be visualised by standard tools.

6.2.2 Infrastructure

All of the studies directly addressing modelling approaches are able to model the digital layer of the infrastructure. About half of them model the physical infrastructure as well. Only few studies reflect the social layer of the system infrastructure. In addition, most of the studies model only the attacker who traverses the system acquiring knowledge and/or access on the way. Our model expresses all three layers, and it is also able to adapt to the existing approaches.

6.2.3 Assets and Containment

All models support assets on the different levels considered. Some studies only consider information, all other include physical and to some extent digital artefacts.

Containment is only addressed by a few studies. The studies that are able to model an actor at a location, are also able to model the nesting of objects. Some studies provide a different notion of containment in the sense of annotation/quantification properties, *e.g.*, hosts containing vulnerabilities.

Our model can represent assets such as data and items. The model represents containment through location attributes; containment can be arbitrarily deep.

6.2.4 Processes, Actions and Behaviour

All studies agree on the definition of a process as a sequence of steps. Most studies only mention the existence of processes as part of the model. However, some models provide explicit support for processes.

In most studies, actions relate to the digital domain varying from specific actions such as Read and Out to "all computing" in general.

Behaviour relates to the social domain, where human actors perform actions within the model. Human behaviour involves all attacker behaviour that is needed to achieve a goal and more specific behaviour like moving between locations or starting a process on a computer. Using human behaviour as a general, non-restricted concept provides flexibility. On the other hand, a restricted set of actions enables formal treatment and analysis. A trade-off should allow the freedom to model human behaviour in a proper way, and also be able to be formally checked.

Our system model also has a natural support for processes, and supports actions and behaviour through the underlying process calculus.

6.2.5 Actors

Most of the studies focus on insiders as they have better access and knowledge. Some of the studies do not distinguish between insiders and outsiders. Few studies do not even distinguish between humans and non-humans in the modelling phase. In real life attackers can collaborate; only few of the studies are able to represent this. One study does not even model actors or a potential attacker, but analyses the context for possible information leakage without specific attack scenario involving an actor. Only two studies model attacks carried out by more than one attacker, *i.e.*, by collaborating attackers.

In our model we do not distinguish between insiders and outsiders. They could differ in their knowledge about the organisation. However, at this stage we do not support collaborating actors.

6.2.6 Policies

Most of the studies take into consideration only low level policies in terms of accessibility/reachability. While considering high level (organisational policies) is essential, it could also be problematic in case of inconsistencies between low and high level policies. One study pays close attention to this problem and derives attack scenarios from analysing the policies on different levels.

Our policy-specification language enables the reasoning about policies and their relationship; especially contradictions between and holes in policies are of interest, since they may enable attacks.

6.2.7 Quantitative Measures

A number of studies support quantitative annotations to annotate the model and attack. Those that are most frequently supported are: probability that an attack will succeed and the costs of an attack. The supported annotations are mainly properties of an attacker or actions, *e.g.*, required skill and risk of detection, however there are also some organisational properties, *e.g.*, impact of an attack.

Our model supports quantitative annotations through unique identifiers for all model elements. These identifiers enable the mapping from elements to properties through the knowledge-base.

6.2.8 Attacks, Vulnerabilities and Countermeasures

Only few of the studies provide vulnerabilities as an input to the model. Usually they are described by domain experts and are based on previous attacks. The most frequently used attack representation is a sequence of actions. While it gives a high overview and is easy to understand by non-experts, it has a flat structure and thus does not provide many details. The other popular representation is attack trees, presenting threats in a hierarchical structure; however, they loose the sequential notion. Another disadvantage of attack trees (or in some studies referred to as attack patterns) is that they cannot reflect interactions between different attacks.

The model described in this dissertation represents vulnerabilities and countermeasures through qualitative properties in the knowledge-base.

6.3 Concluding Remarks

In this section we discuss some questions regarding granularity and properties of the system model with regards to the cloud case study.

Similarly to the discussion in Section 5.5.2, the granularity of the model depends on the modeller. For simplicity, we have not introduced, for example, the power supply or cooling in the system model of the cloud scenario. However, doing so would result in additional alternative attack vectors.

There is a very subtle borderline when it comes to whether an asset should be represented as data or as an item. For example, in Section 6.1.2 we have modelled each of the virtual machines as a tuple $(VM1, hypervisor1)$ which expresses a virtual machine as an item located at a location $hypervisor1$. In reality, a virtual machine is not tangible and not persistent (when in memory). On the other hand, if we had modelled the virtual machine as data, then we would have lost the notion of tuple space, which is necessary for the processes. We remind the reader that in process calculus the process actions operate on tuple spaces (except for the creation which works on nodes).

At the same time, in Section 3.6 we have also modelled a payment card as a tuple $(card, Charlie)$, *i.e.*, expressing it as an item at an actor location $Charlie$. Even though it is modelled exactly in the same way as the virtual machine, in reality the card is both tangible and persistent.

Another example is the file from the scenario presented in Section 6.1. The file on $VM1$ is modelled as data (formally presented as a tuple $(fileX, 42)$). While being digital, the information in the file is not tangible and persistent. However, assume we have the following process definition:

$$\begin{array}{l}\textbf{in}\,(\text{“}print\text{”}, !data)\,@VM1.\\ \textbf{out}\,(data)\,@room\end{array}$$

In the first part of the process, when acquiring the data from the virtual machine $VM1$, *data* itself is not tangible and not persistent. However, the moment the data is printed on paper, it becomes both tangible and persistent. We should note that in this case it might become problematic for the attack generation technique we propose.

Drilling down into more details, we would need to express, for example, a process representing establishment of an ssh tunnel. For this the user would need more advanced modelling. In principle there is a potential to use macros and define patterns of processes. While this can be a future direction for investigation, it is also slightly slipping away from our scope.

Chapter 7

Conclusion

In this dissertation we have explored how to combine social and technical vulnerabilities of organisations by means of applying a system modelling approach to socio-technical systems. We have devised a process-algebraic language that naturally facilitates an analytical identification of attacks in the form of attack trees. Our initial thesis was that

applying a formal modelling approach to the study of socio-technical systems allows to develop algorithms for the automated identification of complex attacks that exploit the interplay between technological and social vulnerabilities.

Below we summarise the contributions of this dissertation, showing how our developments support the claim above. Then we describe how this work is placed in the wider landscape of the TRE$_{\text{S}}$PASS project. Finally, we highlight directions for future investigation.

Presenting an organisation as a socio-technical system. In many of the studies we examined in our literature review, attacks are mostly concerning the technical aspect, neglecting the physical infrastructure and the human aspect. Due to the complex nature of organisational attacks, it is essential to encompass all the aforementioned components in a single model to be able to properly identify threats that are neither purely technical, nor purely social. Therefore,

our work rests on the basic premise, that organisations should be modelled as socio-technical systems.

Socio-technical systems can be expressed by formal languages. We introduced some modifications to a variant of the Klaim language, which gave us the possibility to formalise socio-technical systems. We also defined our policy-specification language which is simple, yet highly expressive. By having socio-technical systems expressed by a formal language, we enable the possibility to conduct automated threat analysis by using policy invalidation.

Organisational vulnerabilities can be identified analytically. Using formal system models, our approach systematically identifies the necessary steps to invalidate policies that are supposed to hold in the modelled system. We formalise attack tree generation *including* human factors. While the notion of "human factor" could be stronger, our approach is flexible enough to support all kinds of human factors that can be instantiated once an attack has been identified. The generated attacks include all relevant steps, from detecting the required assets, obtaining them, and any credentials needed to do so, and finally performing actions that are prohibited in the system. The generated attacks are precise enough to illustrate the threat, and they are general enough to hide the details of individual steps.

Analytical threat analysis is more exhaustive than brainstorming. Current risk assessment methods are heavily dependent on the defender's ability to conceive and define potential attack opportunities in order to prevent them. Even though this approach has proven to work in not too volatile worlds, with the dynamic attack landscape nowadays, this approach is often too slow and over-challenges the human imaginative capabilities. With the framework we have proposed, it is no longer a strong requirement to have security experts manually perform threat analysis based on experience.

There is a way to overcome the gap between academia and industry. We have successfully evaluated our approach on two case studies provided by industry. While both case studies are in different domains, our proposed technique is expressive and flexible enough to fully address the challenges of identifying socio-technical vulnerabilities.

7.1 The socio-technical security model in the context of the TREsPASS project

Risk assessment methods demand tool support to predict, prioritise and prevent complex attack scenarios analytically. This is exactly the core of the TREsPASS project [TREsPASS]. In this dissertation we have presented a modelling framework based on socio-technical security system models, which have a central role in the technical part of the TREsPASS project. They constitute the interface between the organisation being modelled and the analysis and visualisation tools developed by the other partners in TREsPASS.

Being part of the TREsPASS project, the research in this thesis was also aligned with the overall goal of the project, and more precisely synchronised with the parts closely related to the model.

The work described in this thesis is an essential piece of the TREsPASS "puzzle", where the risk assessment framework as a whole is the one with the real value. In this section we outline and discuss the relation of our modelling approach together with the attack generation technique we have presented in this dissertation to the TREsPASS project.

The methodology proposed in this dissertation consists of three major phases:

- describing an organisation as a socio-technical security system model
- with a given goal, identify all the attack vectors from an initial state of the model
- gathering the attack vectors in an attack tree

Figure 7.1 shows the relations of the different major components in the TREsPASS project, where the system model serves as an entry point to the complete risk assessment framework. A good overview of the TREsPASS project can also be found at [PWP15]. The work in this thesis fully covers the *Model* section in Figure 7.1 as well as partially the *attack identification* in terms of generating attack trees based on the system model and a certain goal.

As mentioned earlier, the generated attacks do not consider any attacker properties like skills, resources or motivation. Instead, different attacker profiles are introduced based on agent modelling [Cas07; Ros10]. The usage of the attacker profiles simplifies the attack identification given the system model (as opposed to, for example, defining the actors' behaviour as a component in the system

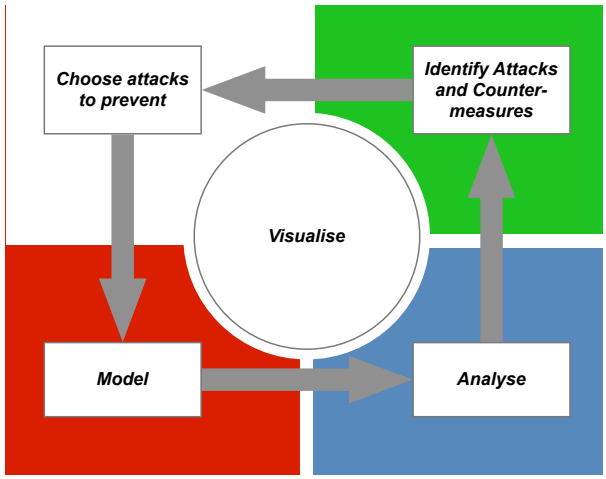


Figure 7.1: The machinery of the TRESPASS project

model). As a consequence it also enables the simplification of the defenders. It is easier to identify the relevant countermeasures according to the different threat levels.

A risk assessment framework without countermeasure analysis would be useless. The automation of the attack trees generation allows for the further analysis and identification of relevant countermeasures. The analysis phase in Figure 7.1 estimates the impact of the attack vectors. Various quantitative analysis techniques have been proposed by TRESPASS partners, assigning quantitative metrics to the basic actions of an attack tree and propagating them to the root of the tree.

Based on the results of the aforementioned analyses, attack vectors are then prioritised. We believe that a fully-automated framework is not possible and it is actually not even needed. The user should intervene and decide which actions to take in order to prevent given vulnerabilities based on the resources and the cost of introducing them. Estimation of the pay-off of introducing a certain countermeasure is highly context-dependent and requires human interaction. When decisions are made, the model can then be changed accordingly and a rerun of the analysis would reveal the updated state of the organisational vulnerabilities.

As mentioned earlier in Section 3.8, the granularity on each of the modelling layers can be adjusted depending on the kind of vulnerabilities the modeller is interested in. Different classes of vulnerabilities require different levels of details with regards to the modelling. The precision of the modelling directly affects

the complexity of the attack tree generation.

To simplify the attack tree generation, we predefine certain standard sub-trees expressing standard sub-goals. These sub-trees are not part of the automatic attack tree generation, but instead are stored in attack pattern libraries and later plugged-in [PWP15].

7.2 Future Directions

For a more detailed discussion, the reader is referred to the concluding remarks following each of the chapters. In this section, on a higher-level, we highlight some topics which would be beneficial for this dissertation.

A major restriction of the work presented in this dissertation is that the system model is analysed and different attack vectors are derived from the model, under the assumption that an entire attack is carried out by a single attacker. In reality, it is often the case that an attack is performed by collaborating attackers. As a result, attacks, which according to our current proposal might seem infeasible after the qualitative analysis of the attack trees, would suddenly turn out to have been dangerously neglected. Therefore, it is worthwhile to enable collaborating attackers and be able to analyse the impact of an attack considering combinations of skills, knowledge, etc. (*i.e.* attackers' resources and capabilities) of multiple attackers.

Obviously, building a fully secured system cannot be achieved, but combining our approach with the further analysis of the derived attack trees [AN15] together with the "plug-and-play" attacker profiles [LWS14; Per14], establishes bridges and invites for setting priorities and introducing actual countermeasures. In recent studies the notion of countermeasures is introduced on a lower level, generating *attack-defence trees* directly from the system model [Kor+12]. Since this is another feasible approach, which potentially gives more flexibility (where having iterations of changing the model and re-running the analysis remains sensible), and considering, that the pure modelling part of our approach is independent from the attack tree generation (*i.e.,* it is rather flexible), it would be interesting to investigate under which assumptions attack-defence trees could be generated from our formal system model.

As mentioned earlier, there are studies exploring the reasoning about policy inconsistencies, especially between policies on different levels (for example, local access control policies and policies defining the organisational behaviour of the employees). The design of our policy-specification language enables such

reasoning, and it is definitely a direction worth exploring, since it can possibly identify attacks originating from inconsistencies [PDP13b].

The target group has always been security practitioners, who should be able to apply the machinery in practice, thus helping themselves in the cumbersome and highly error prone manual attack identification process. For this, it is desirable to provide a user-friendly implementation by enhancing our current implementation with having a standard user interface for model development and customised visualisation of the potential attacks.

Last but not least, it would be interesting to enable a qualitative comparison with other risk assessment approaches.

Appendix A

The XML model structure

In this Appendix we describe in more details the XML format used for the implementation. The description is based on [D3.4.1]. We start with the structure of the scenario.

Scenario Each scenario is stored in different XML file which has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario
    xmlns="..."
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="..."
    id="..."
    author="..."
    date="..."
    version="...">
   <model>test/TREsPASS_model.xml</model>
   <goal>
       ...
   </goal>
</scenario>
```

Here, `<model>` refers to a model to be analysed, and `<goal>` is a policy, which is expected to be enforced system-wide.

Model A model mentioned in the scenario has the following overall structure. Each of the components are further described below.

```
<?xml version="1.0" encoding="UTF-8"?>
<system
    xmlns="..."
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="..."
    author="..."
    date="..."
    version="...">
  <title>...</title>
  <locations>...</locations>
  <edges>...</edges>
  <assets>...</assets>
  <actors>...</actors>
  <roles>...</roles>
  <predicates>...</predicates>
  <policies>...</policies>
  <processes>...</processes>
</system>
```

The general structure of the model description looks as follows: The title element is a string presenting the name of the model. For example:

```
<title>IPTV model</title>
```

Locations The `<locations>` section describes the locations associated with the model. Each location has an identifier (so that policies can be associated with locations) and a domain. There are two types of domains: `physical` and `network`, with the idea that capability to move around in one domain does not automatically mean moving around in the other (*e.g.,* humans can not move in a network and programs/processes in the physical world).

Example:

```
<locations>
  <location id="Home" domain="physical" />
  <location id="Computer C" domain="network" />
</locations>
```

Edges The <edges> section describes existing connections between the locations. Each <edge> will have a <source> and <target>. <source> and <target> may be either an explicitly defined <location> or <item> from the <assets> declarations.

Example:

```
<edges>
  <edge>
    <source>home</source>
    <target>city</target>
  </edge>
  <edge>
    <source>PC</source>
    <target>Voting_server</target>
  </edge>
</edges>
```

Assets The <assets> section describes the items and data relevant to the model. <item> is something that one can physically have and/or move around (payment card, PC, money). <data> represents a piece of knowledge and/or digital asset (PIN, password, cryptographic key, value of a vote). Computer programs are not typically assets, but rather <processes> running at some items (like PC) or locations (like network servers that one can not physically access).

Assets also have (initial) locations associated with them. These are the places where the assets are expected to be located. It is neither necessary, nor advisable to try to express all the possible locations in the model, since this is handled by the attack generation procedure, taking into account the <edges> between locations and the respective <policies>. The <atLocations> element may contain a list of locations separated by white spaces.

It is important to note that the <atLocations> tag may also list other objects than just the ones declared as <locations>. <actors> and <items> may also serve as locations for other items or data (*i.e.*, containment property).

Every asset needs to have a unique id, since there may be various assets with the same name (card, PIN), so the id can be used to distinguish between them. The asset name can also be interpreted as its type. In general, attribute id is used when a unique object reference is needed, and attribute name is used when different objects may have the same reference.

Example:

```
<assets>
  <item name="Card" id="x001">
    <atLocations>Home</atLocations>
  </item>
  <data name="PIN" id="x009">
    <atLocations>Alice Card</atLocations>
  </data>
</assets>
```

Actors The `<actors>` section describes the actors relevant to the model. An `<actor>` may be `<atLocations>` which is again a whitespace-separated list of locations. And again, this list is meant to be just the initial list of locations (of which there may quite possibly be just one) which the attack generation tool must be able to take care of moving the actors around in the model.

Example:

```
<actors>
  <actor id="Alice">
    <atLocations>home</atLocations>
  </actor>
  <actor id="Charlie">
    <atLocations>city</atLocations>
  </actor>
</actors>
```

Roles Many access control policies are actually stated in the terms of roles. To capture that concept, `<roles>` may also be declared in the model. A role has a unique `id` and possibly several persons (referenced by `<actorID>`) being assigned this role.

Example:

```
<roles>
  <role id="customer">
    <actors>
      <actorID>Alice</actorID>
    </actors>
  </role>
  <role id="care taker">
    <actors>
      <actorID>Fred</actorID>
      <actorID>Charlie</actorID>
    </actors>
  </role>
</roles>
```

Predicates The `<predicates>` section describes different predicates that hold between the actors. Predicates in the model are not predefined, so the attack generation will just generate a node stating "Fulfil the predicate . . . " and further expansion of this node must come from the Attack Pattern Library.

Example:

```
<predicates>
  <predicate name="trusts">
    <parameterID>Alice</parameterID>
    <parameterID>care taker</parameterID>
  </predicate>
</predicates>
```

Note that the parameters of a predicate may have different types (`Alice` is an `<actor>`, whereas `care taker` is a `<role>`), so attack generation has to take this into account.

Policies The `<policies>` section defines policies that enable certain actions or access to some locations, assuming certain preconditions are met.

A policy consists of three parts: `<credentials>`, `<enabledActions>` and `<atLocations>` declarations. `<credentials>` describe preconditions that have to be met in order for the policy to allow an action at certain locations. A credential may comprise of one or several of the following:

- be at a certain location,
- be a certain actor,
- have a certain role,
- possess a certain asset (either item of data),
- fulfil a predicate.

Enabled actions may be one of the following:

- `i`: input data,
- `r`: read data,
- `o`: output data,

- `m`: move an actor or a process,
- `e`: execute a process.

The `<atLocations>` tag refers to the location where this policy is applied. It is important to understand the difference between the `<credLocation>` and `<atLocations>` tags. The former refers to a precondition expressed in the form of being at a location (*e.g.*, in the room that has a door), the latter refers to the actual location of policy enforcement (*e.g.*, the door). It is likely that `<atLocations>` will often be referring to "virtual" locations that do not correspond directly to our everyday understanding of the term 'location'.

`<credItem>` and `<credPredicate>` may have arguments.

`<credItem>` may have either one or several `<credData>` or `<credItem>` instances attached to it. `<credData>` may either refer to an explicit `<value>` or `<variable>`.]

Similarly, `<credPredicate>` may have one or more `<argument>` tabs which in turn may either contain an explicit `<value>` (which must refer to a valid `id`) or a variable.

Variables are unified by name across different occurrences, and hence provide a way of binding different model components to each other.

Example:

```
<policies>
  <policy>
    <credentials>
      <credLocation id="..."/>
      <credActor id="..."/>
      <credRole id="..."/>
      <credItem name="card">
        <argument>
          <credData name="pin">
            <variable>X</variable>
          </credData>
        </argument>
      </credItem>
      <credData name="pin">
        <variable>X</variable>
      </credData>
      <credPredicate name="isActor">
```

```
        <argument>
          <value>Margrethe</value>
        </argument>
      </credPredicate>
    </credentials>
    <enabledActions>...</enabledActions>
    <atLocations>...</atLocations>
  </policy>
</policies>
```

Processes The `<processes>` section describes the processes running in the model. A `<process>` runs at (a) certain location(s). After receiving a `<signal>` (say, as an output of an action run after a successful policy evaluation) it waits for some `<inputs>` and after receiving them, gives the corresponding `<outputs>`.

Example:

```
<processes>
  <process id="Encryption">
    <atLocations>PC</atLocations>
    <signal>
      Encrypt
    </signal>
    <inputs>
      <input datatype="vote">X</input>
      <input>Serv_Pub_Key</input>
      <input>owner</input>
    </inputs>
    <outputs>
      <output>Encrypted_vote</output>
      <output>owner</output>
    </outputs>
  </process>
</processes>
```

Appendix B

The tool input (XML) /output (AT)

B.1 XML input

```
<?xml version="1.0" encoding="UTF-8"?>
<system
    xmlns="https://www.trespass-project.eu/schemas/TREsPASS_model"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.trespass-project.eu/schemas/TREsPASS_model.xsd
    author="Marieta G ivanova"
    date="11-01-2015"
    version="2.0">
  <title>Cloud infrastructure model</title>
  <locations>
    <!-- rooms in building -->
    <location id="Outside"/>
    <location id="DoorExternal"/>
    <location id="Hallway"/>
    <location id="DoorInternal"/>
    <location id="RoomInternal"/>
    <location id="WindowInternal"/>
    <location id="DoorDatacenter"/>
```

```
    <location id="RoomDatacenter"/>
    <location id="WindowDatacenter"/>
  </locations>
  <edges>
    <!-- connectivity building -->
    <edge>
      <source>Outside</source><target>DoorExternal</target>
    </edge>
    <edge>
      <source>DoorExternal</source><target>Hallway</target>
    </edge>
    <edge>
      <source>Hallway</source><target>Outside</target>
    </edge>
    <edge>
      <source>Hallway</source><target>DoorInternal</target>
    </edge>
    <edge>
      <source>DoorInternal</source><target>RoomInternal</target>
    </edge>
    <edge>
      <source>RoomInternal</source><target>Hallway</target>
    </edge>
    <edge directed="false">
      <source>RoomInternal</source><target>WindowInternal</target>
    </edge>
    <edge directed="false">
      <source>Outside</source><target>WindowInternal</target>
    </edge>
    <edge directed="false">
      <source>RoomInternal</source><target>DoorDatacenter</target>
    </edge>
    <edge directed="false">
      <source>DoorDatacenter</source><target>RoomDatacenter</target>
    </edge>
    <edge directed="false">
      <source>WindowDatacenter</source><target>RoomDatacenter</target>
    </edge>
    <edge directed="false">
      <source>Hallway</source><target>WindowDatacenter</target>
    </edge>
    <!-- connectivity cyber -->
    <edge directed="false">
      <source>Laptop</source><target>Switch1</target>
```

```
    </edge>
    <edge directed="false">
      <source>Switch1</source><target>Server1</target>
    </edge>
    <!-- connectivity switch -->
    <edge directed="false">
      <source>Server1</source><target>Switch2</target>
    </edge>
    <edge directed="false">
      <source>Server2</source><target>Switch2</target>
    </edge>
  </edges>
  <assets>
    <item name="idcard" id="x002">
      <atLocations>Sydney</atLocations>
    </item>
    <item name="idcard" id="x003">
      <atLocations>Terry</atLocations>
    </item>
    <item name="idcard" id="x004">
      <atLocations>Ethan</atLocations>
    </item>
    <item name="idcard" id="x005">
      <atLocations>Finn</atLocations>
    </item>
    <!-- cyber infrastructure -->
    <item name="Laptop" id="Laptop">
      <atLocations>RoomInternal</atLocations>
    </item>
    <item name="Switch1" id="Switch1">
      <atLocations>RoomInternal</atLocations>
    </item>
    <!-- cyber infrastructure server 1-->
    <item name="Server1" id="Server1">
      <atLocations>RoomDatacenter</atLocations>
    </item>
    <item name="Switch2" id="Switch2">
      <atLocations>RoomDatacenter</atLocations>
    </item>
    <!-- cyber infrastructure server 2-->
    <item name="Server2" id="Server2">
      <atLocations>RoomDatacenter</atLocations>
    </item>
    <item name="VM1" id="VM1">
```

```
    <atLocations>Server1</atLocations>
  </item>
  <data name="pin" id="x006" value="1">
    <atLocations>Sydney x002</atLocations>
  </data>
  <data name="pin" id="x007" value="2">
    <atLocations>Terry x003</atLocations>
  </data>
  <data name="pin" id="x008" value="3">
    <atLocations>Ethan x004</atLocations>
  </data>
  <data name="pin" id="x009" value="4">
    <atLocations>Finn x005</atLocations>
  </data>
  <data name="owner" id="x010" value="Sydney">
    <atLocations>x002</atLocations>
  </data>
  <data name="owner" id="x011" value="Terry">
    <atLocations>x003</atLocations>
  </data>
  <data name="owner" id="x012" value="Ethan">
    <atLocations>x004</atLocations>
  </data>
  <data name="owner" id="x013" value="Finn">
    <atLocations>x005</atLocations>
  </data>
  <data name="fileX" id="fileX" value="42">
    <atLocations>VM1 Laptop Switch1 Switch2</atLocations>
  </data>
  <data name="password" id="pwdethan" value="1234">
    <atLocations>Ethan</atLocations>
  </data>
  <data name="password" id="pwdsydney" value="2345">
    <atLocations>Sydney</atLocations>
  </data>
  <data name="password" id="admpwdsydney" value="3456">
    <atLocations>Sydney</atLocations>
  </data>
</assets>
<actors>
  <actor id="Finn">
    <atLocations>Outside</atLocations>
  </actor>
  <actor id="Ethan">
```

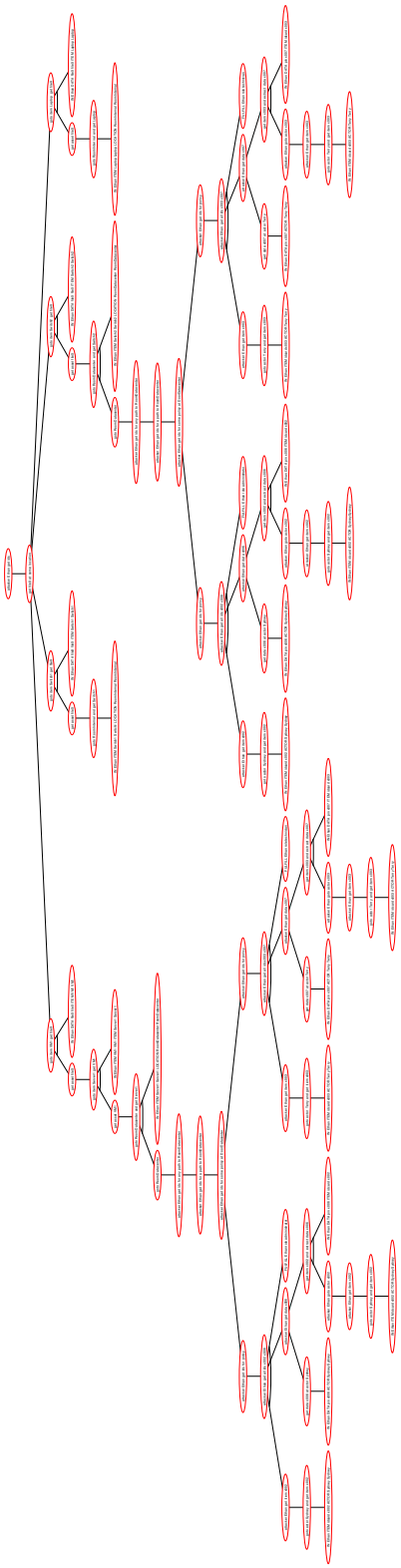
```
    <atLocations>Outside</atLocations>
  </actor>
  <actor id="Sydney">
    <atLocations>Outside</atLocations>
  </actor>
  <actor id="Terry">
    <atLocations>Outside</atLocations>
  </actor>
</actors>
<policies>
  <policy id="p001">
    <credentials>
      <credItem name="idcard">
        <credData name="pin">
          <variable>Y</variable>
        </credData>
          </credItem>
      <credData name="pin">
        <variable>Y</variable>
      </credData>
    </credentials>
    <enabled><move /></enabled>
    <atLocations>DoorExternal</atLocations>
  </policy>
  <policy id="p002">
    <credentials>
      <credItem name="idcard">
        <credData name="owner">
          <variable>X</variable>
        </credData>
        <credData name="pin">
          <variable>Y</variable>
        </credData>
          </credItem>
      <credPredicate name="role">
        <value>employee</value>
          <variable>X</variable>
      </credPredicate>
      <credData name="pin">
        <variable>Y</variable>
      </credData>
    </credentials>
    <enabled><move /></enabled>
    <atLocations>DoorInternal</atLocations>
```

```
</policy>
<policy id="p003">
  <credentials>
    <credItem name="idcard">
      <credData name="owner">
        <variable>X</variable>
      </credData>
             <credData name="pin">
        <variable>Y</variable>
      </credData>
    </credItem>
    <credPredicate name="role">
      <value>administrator</value>
      <variable>X</variable>
    </credPredicate>
    <credData name="pin">
      <variable>Y</variable>
        </credData>
  </credentials>
  <enabled><move /></enabled>
  <atLocations>DoorDatacenter</atLocations>
</policy>
<policy id="p004">
  <credentials>
    <credItem name="idcard">
      <credData name="owner">
        <variable>X</variable>
      </credData>
      <credData name="pin">
        <variable>Y</variable>
      </credData>
        </credItem>
    <credPredicate name="role">
      <value>technician</value>
      <variable>X</variable>
    </credPredicate>
    <credData name="pin">
             <variable>Y</variable>
        </credData>
  </credentials>
  <enabled><move /></enabled>
  <atLocations>DoorDatacenter</atLocations>
</policy>
<policy id="p005">
```

```
      <credentials>
        <credData name="password">
          <value>1234</value>
        </credData>
      </credentials>
      <enabled><in /></enabled>
      <atLocations>VM1 Laptop</atLocations>
    </policy>
  </policies>
  <processes>
  </processes>
  <predicates>
    <predicate id="role" arity="2">
      <value>administrator Sydney</value>
      <value>technician Terry</value>
      <value>accountant Finn</value>
      <value>investigator Ethan</value>
      <value>employee Finn</value>
      <value>employee Ethan</value>
      <value>employee Sydney</value>
      <value>employee Terry</value>
    </predicate>
    <predicate id="isPassword" arity="3">
      <value>vm1 Ethan pwdethan</value>
      <value>laptop Ethan pwdethan</value>
      <value>laptop Sydney pwdsydney</value>
      <value>hypervisor1 Sydney admpwdsydney</value>
    </predicate>
  </predicates>
</system>
```

B.2 Generated Attack Tree

In the following pages, we show the attack tree for the actor Ethan as a potential attacker. Unfortunately, the labels of the nodes are not readable in the hard copy, but they can be seen in the digital version of this dissertation together with the refined version of the attack tree.



Appendix C

The Isabelle Theory

```
theory enables
imports Main
begin
datatype action = get | move | eval |put
datatype actor  = Actor string
type_synonym policy = "((actor ⇒ bool) × action set)"
datatype location = Location nat
datatype node = NA actor | NL location
datatype graph = Graph "(node × node)set"
datatype infrastructure = Infrastructure "graph" "location ⇒ policy set"
  "actor ⇒ bool"
primrec act :: "actor ⇒ string"  where  "act(Actor n) = n"
primrec loc :: "location ⇒ nat"  where  "loc(Location n) = n"
primrec gra :: "graph ⇒ (node × node)set"  where  "gra(Graph g) = g"
definition nodes_graph :: "graph ⇒ node set"
  where "nodes_graph g ≡ { x. ∃ y. (x,y) ∈ gra g) ∨ (y,x) ∈ gra g }"
definition actors_graph :: "graph ⇒ actor set"
  where  "actors_graph g ≡ {x. ∃ y. NA(Actor y) : nodes_graph g ∧ x = Actor y}"
definition locs_graph :: "graph ⇒ location set"
  where  "locs_graph g ≡ {x. ∃ y. NL(Location y) ∈ nodes_graph g ∧ x =  Location y}"
primrec graphI :: "infrastructure ⇒ graph"
  where "graphI (Infrastructure g d c) = g"
primrec delta :: "[infrastructure, location] ⇒ policy set"
  where "delta (Infrastructure g d c) = d"
primrec tspace :: "[infrastructure, actor] ⇒ bool"
  where "tspace (Infrastructure g d c) = c"
primrec actorsI :: "infrastructure ⇒ actor set"
  where "actorsI (Infrastructure g d c) = actors_graph g"
definition enables :: "[infrastructure, location, actor, action] ⇒ bool"
```

```
  where
  "enables I l a a' ≡  ∃ (c,e) ∈ delta I l. a' ∈ e ∧ (tspace I a ⟶ c(a))"
definition ID :: "[actor, string] ⇒ bool"
  where "ID a s ≡ (act a = s)"
consts data_expr :: "['a × 'b] ⇒ bool"
consts card :: "['a × 'b] ⇒ bool"
consts role :: "actor × string ⇒ bool"
definition inpred :: "[bool, bool] ⇒ bool" ("_ inp _" [50,50]50)
  where "Q inp P ≡ (P ⟶ Q)"
(* Invalidation formula *)
definition inv_formula_ex :: "infrastructure ⇒ bool"
where "inv_formula_ex I ≡
  ∀ x ∈ actorsI I. ∀ y ∈ actorsI I.
  (x ≠ y ∧ (role(x, ''employee'') inp tspace I x) ∧
    ((card(''owner'',y) ∧ role(y,''customer'')) inp tspace I y))
  ⟶ ¬(∃ t. enables I t x get)"

(* example infrastructure 1 = bank computer, 2 = home *)
definition ex_graph :: "graph"
where "ex_graph ≡ Graph (NA (Actor ''Charly''), NL (Location 1)),
    (NL (Location 2), NL (Location 1)), (NA (Actor ''Alice''), NL (Location 2))"

definition ex_policy_bank :: "location ⇒ policy set"
where "ex_policy_bank ≡ (λ x. if x = Location 1 then
  {(λ x. ∃ X :: nat. card(''pin'',X) ∧ data_expr(''pin'',X), {get})} else {})"
definition ex_creds :: "actor ⇒ bool"
where "ex_creds ≡ (λ x. if x = Actor ''Charly'' then
    card(''pin'',42::nat) ∧ data_expr(''pin'',42::nat) ∧
      role(Actor ''Charly'',''employee'')
  else (card(''owner'',Actor ''Alice'') ∧ role(Actor ''Alice'',''customer'')))"
definition scenario_one :: "infrastructure"
where "scenario_one ≡ Infrastructure ex_graph ex_policy_bank ex_creds"
lemma ex_inv : "¬ inv_formula_ex scenario_one"
apply (simp add: inv_formula_ex_def scenario_one_def)
apply (rule_tac x = "Actor ''Charly''" in bexI)
apply (rule conjI)
apply (rule_tac x = "Actor ''Alice''" in bexI)
apply (simp add: inv_formula_ex_def scenario_one_def
        ex_creds_def ex_policy_bank_def ex_graph_def inpred_def)
apply (simp add: ex_graph_def actors_graph_def nodes_graph_def)
apply (rule_tac x = "Location 1" in exI)
apply (simp add: enables_def ex_creds_def ex_policy_bank_def ex_graph_def inpred_def)
apply (rule impI)
apply (rule_tac x = "42" in exI)
by (simp add: ex_graph_def actors_graph_def nodes_graph_def)+
end
```

Bibliography

[82] *The American Heritage Dictionary*. Houghton Mifflin, 1982, 1568 pages.

[AK12] Bilal Al Sabbagh and Stewart Kowalski. "ST(CS)2 - Featuring socio-technical cyber security warning systems". In: *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*. IEEE, June 2012, pp. 312–316.

[AN15] Zaruhi Aslanyan and Flemming Nielson. "Pareto Efficient Solutions of Attack-Defence Trees". In: *Proceedings of the 4th International Conference on Principles of Security and Trust POST 2015*. 2015, pp. 95–114.

[Asl+15] Zaruhi Aslanyan et al. "Modeling and analysing socio-technical systems". In: (2015).

[Bis+10] Matt Bishop et al. "A Risk Management Approach to the "Insider Threat"". In: *Insider Threats in Cyber Security*. Ed. by Christian W. Probst et al. Vol. 49. Advances in Information Security. Springer-Verlag, 2010, pp. 115–137.

[Bis02] Matt Bishop. *Computer Security: Art and Science*. Boston USA: Addison-Weley, 2002, p. 1136.

[BLP02] Lorenzo Bettini, Michele Loreti, and Rosario Pugliese. "An infrastructure language for open nets". In: *Proceedings of the 2002 ACM symposium on Applied computing*. SAC '02. Madrid, Spain: ACM, 2002, pp. 373–377.

[Boe+14] Jaap Boender et al. "Modeling human behaviour with higher order logic: insider threats". In: *Socio-Technical Aspects in Security and Trust (STAST), 2014 Workshop on*. IEEE. 2014, pp. 31–39.

[BS04] Moritz Y Becker and Peter Sewell. "Cassandra: Flexible trust management, applied to electronic health records". In: *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*. IEEE. 2004, pp. 139–154.

[Bul+] Jan-Willem Bullee et al. "Systematic Literature Review on Socio-Technical Security Models". In: (). In preparation for submission.

[Cas07] Timothy Casey. "Threat Agent Library Helps Identify Information Security Risks". In: *Intel White Paper* (2007).

[CC12] Liang Chen and Jason Crampton. "Risk-Aware Role-Based Access Control". English. In: *Security and Trust Management*. Ed. by Catherine Meadows and Carmen Fernandez-Gago. Vol. 7170. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 140–156.

[Cer01] Iliano Cervesato. "The Dolev-Yao Intruder is the Most Powerful Attacker". In: *Proc. of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*. IEEE Computer Society Press, June 2001, pp. 16–19.

[CHM13] Jason Crampton, Michael Huth, and Charles Morisset. *Policy-Based Access Control from Numerical Evidence*. Tech. rep. 2013/6. Retrived on 26 October 2015. London, SW7 2AZ, England: Imperial College London, Department of Computing, Oct. 2013.

[CK06] Jason Crampton and Hemanth Khambhammettu. "Delegation in Role-Based Access Control". English. In: *Computer Security – ESORICS 2006*. Ed. by Dieter Gollmann, Jan Meier, and Andrei Sabelfeld. Vol. 4189. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 174–191.

[CK08] Jason Crampton and Hemanth Khambhammettu. "Delegation in role-based access control". In: *International Journal of Information Security* 7.2 (2008), pp. 123–136.

[Clo10] Cloud Security Alliance. *Top Threats to Cloud Computing V1.0*. https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf. 2010.

[Col09] Carl Colwill. "Human factors in information security: The insider threat - Who can you trust these days?" In: *Information Security Technical Report* 14.4 (2009), pp. 186–196.

[CR06] Sudip Chakraborty and Indrajit Ray. "TrustBAC: Integrating Trust Relationships into the RBAC Model for Access Control in Open Systems". In: *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*. SACMAT '06. Lake Tahoe, California, USA: ACM, 2006, pp. 49–58.

[D1.2.2] The TRE$_S$PASS Project, D1.2.2. *Final policy-specification language*. Deliverable D1.2.2. 2015.

[D1.3.2] The TRE$_S$PASS Project, D1.3.2. *Extensibility of socio-technical security models*. Deliverable D1.3.2. 2015.

[D3.4.1] The TRE$_S$PASS Project, D3.4.1. *Attack generation from socio-technical security models*. Deliverable D3.4.1. 2014.

[DC04] Boris Dragovic and Jon Crowcroft. "Information exposure control through data manipulation for ubiquitous computing". In: *Proceedings of the 12th New Security Paradigms Workshop*. 2004, pp. 57–64.

[DC05] Boris Dragovic and Jon Crowcroft. "Containment: from context awareness to contextual effects awareness". In: *Proceedings of 2nd Inernational Workshop on Software Aspects of Context. CEUR Workshop Proceedings*. 2005.

[De +10] Rocco De Nicola et al. "From Flow Logic to static type systems for coordination languages". In: *Science of Computer Programming* 75.6 (2010), pp. 376–397.

[DFP98a] R. De Nicola, G.L. Ferrari, and R. Pugliese. "KLAIM: a kernel language for agents interaction and mobility". In: *Software Engineering, IEEE Transactions on* 24.5 (1998), pp. 315–330.

[DFP98b] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. "KLAIM: A kernel language for agents interaction and mobility". In: *Software Engineering, IEEE Transactions on* 24.5 (1998), pp. 315–330.

[DGP05] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. "Pattern Matching over a Dynamic Network of Tuple Spaces". In: *FMOODS*. 2005, pp. 1–14.

[Dim12] T. Dimkov. *Alignment of Organizational Security Policies – Theory and Practice*. University of Twente, 2012.

[DPH10a] T. Dimkov, W. Pieters, and P. H. Hartel. "Portunes: representing attack scenarios spanning through the physical, digital and social domain". In: *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*. Vol. 6186. Lecture Notes in Computer Science. Paphos, Cyprus: Springer Verlag, 2010, pp. 112–129.

[DPH10b] T. Dimkov, W. Pieters, and P. H. Hartel. "Portunes: representing attack scenarios spanning through the physical, digital and social domain". In: *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10). Revised Selected Papers, Paphos, Cyprus*. Vol. 6186. Lecture Notes in Computer Science. Paphos, Cyprus: Springer Verlag, Mar. 2010, pp. 112–129.

[Dra06] Boris Dragovic. "Casper: Containment-aware security for pervasive computing environments". PhD thesis. 2006.

[ENI09] ENISA. *Cloud Computing Risk Assessment*. Tech. rep. ENISA, 2009.

[ET60] F. E. Emery and E. L. Trist. "Socio-technical Systems." In: *Management Sciences: Models and Techniques vol. 2*. Ed. by C.W. Churchman and M. Verhulst. Pergamon. London, 1960, pp. 83–97.

[FH02] Antonio L Freitas and E Tory Higgins. "Enjoying Goal-Directed Action: The Role of Regulatory Fit". In: *Psychological Science* 13.1 (2002), pp. 1–6.

[FLE09] Virginia N. L. Franqueira, Raul H. C. Lopes, and Pascal van Eck. "Multi-step attack modelling and simulation (MsAMS) framework based on mobile ambients". In: *Proceedings of the 2009 ACM symposium on Applied Computing*. SAC '09. Honolulu, Hawaii: ACM, 2009, pp. 66–73.

[Fra06] X. Franch. "On the quantitative analysis of agent-oriented models". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4001 LNCS (2006), pp. 495–509.

[GP03] Daniele Gorla and Rosario Pugliese. "Resource Access and Mobility Control with Dynamic Privileges Acquisition". English. In: *Automata, Languages and Programming*. Ed. by JosC.M. Baeten et al. Vol. 2719. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 119–132.

[Gra09] I. Grant. *Insiders cause most IT security breaches*. http://www.computerweekly.com/news/1280090551/Insiders-cause-most-IT-security-breaches-study-reveals. [Online; accessed 09-January-2016]. 2009.

[HKT13] Jin Bum Hong, Dong Seong Kim, and Tadao Takaoka. "Scalable Attack Representation Model Using Logic Reduction Techniques". In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (2013), pp. 404–411.

[HMP13] Carly Huth, David Mundie, and Sam Perl. "Toward an Ontology for Insider Threat Research: Varieties of Insider Threat Definitions". In: *Proc. of the 3rd Workshop on Socio-Technical Aspects in Security and Trust (STAST'13)*. 2013.

[HO48] Carl G. Hempel and Paul Oppenheim. "Studies in the Logic of Explanation". In: *Philosophy of Science* 15 (2 Apr. 1948), pp. 135–175.

[HP11] Jeffrey Hunker and Christian W. Probst. "Insiders and Insider Threats An Overview of Definitions and Mitigation Techniques". In: *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications* (2011).

[Hu+14] Vincent C Hu et al. "Guide to Attribute Based Access Control (ABAC) Definition and Considerations". In: *NIST Special Publication* 800 (2014), p. 162.

[Iva+13] Marieta Georgieva Ivanova et al. "Externalizing Behaviour for Analysing System Models". In: *Journal of Internet Services and Information Security* 3.3/4 (2013), pp. 52–62.

[Iva+15a] Marieta Georgieva Ivanova et al. "Attack Tree Generation by Policy Invalidation". In: *Information Security Theory and Practice*. Springer International Publishing, 2015, pp. 249–259.

[Iva+15b] Marieta Georgieva Ivanova et al. "Transforming graphical system models to graphical attack models". In: *Proceedings of the Second International Workshop on Graphical Models for Security (GraMSEC'15) co-located with CSF'15*. IEEE, 2015.

[KBP12] Siwar Kriaa, Marc Bouissou, and Ludovic Pietre-Cambacedes. "Modeling the Stuxnet attack with BDMP: Towards more formal risk assessments". In: *2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS)*. IEEE, Oct. 2012, pp. 1–8.

[Kit04] Barbara Kitchenham. "Procedures for performing systematic reviews". In: *Keele, UK, Keele University* 33 (2004), p. 2004.

[Kor+12] Barbara Kordy et al. "Attack–Defense Trees". In: *Journal of Logic and Computation* (2012). Available at http://logcom.oxfordjournals.org/content/early/2012/06/21/logcom.exs029.

[KOS13] Peter Karpati, Andreas L Opdahl, and Guttorm Sindre. "HARM: Hacker attack representation method". In: *Software and Data Technologies*. Springer, 2013, pp. 156–175.

[KPS14] Barbara Kordy, Ludovic Piétre-Cambacédés, and Patrick Schweitzer. "DAG-based attack and defense modeling: Don't miss the forest for the attack trees". In: *Computer Science Review* 13-14 (2014), pp. 1–38.

[KS12a] Piotr Kordy and Patrick Schweitzer. *ADTool*. http://satoss.uni.lu/members/piotr/adtool. Accessed January 09th, 2016. 2012.

[KS12b] Piotr Kordy and Patrick Schweitzer. *The ADTool Manual*. University of Luxembourg. Oct. 2012.

[KSB11] S. Kandala, R. Sandhu, and V. Bhamidipati. "An Attribute Based Framework for Risk-Adaptive Access Control Models". In: *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*. Aug. 2011, pp. 236–241.

[KSD11] Igor Kotenko, Mikhail Stepashkin, and Elena Doynikova. "Security Analysis of Information Systems Taking into Account Social Engineering Attacks". In: *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing* (2011), pp. 611–618.

[KW13] Florian Kammüller and Christian W. Probst. "Invalidating policies using structural information". In: *2nd International IEEE Workshop on Research on Insider Threats (WRIT'13)*. Co-located with IEEE CS Security and Privacy 2013. IEEE, 2013.

[KW14] Florian Kammüller and Christian W. Probst. "Combining Generated Data Models with Formal Invalidation for Insider Threat Analysis". In: *3rd International IEEE Workshop on Research on Insider Threats (WRIT'14)*. Co-located with IEEE CS Security and Privacy 2014. IEEE, 2014.

[LAC14] R Lee, M Assante, and T Connway. "ICS CP/PE (Cyber-to-Physical or Process Effects) case study paper–German Steel Mill Cyber Attack". In: *Sans ICS, Dec* (2014).

[LBN99] Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. "A Policy Description Language". In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence (AAAI'99/IAAI'99)*. Orlando, Florida, USA, 1999, pp. 291–298.

[LMO15] Gabriele Lenzini, Sjouke Mauw, and Samir Ouchani. "Security analysis of socio-technical physical systems". In: *Computers and Electrical Engineering* (2015). Available online 6 April 2015.

[LWS14] Aleksandr Lenin, Jan Willemson, and Dyan Permata Sari. "Attacker profiling in quantitative security assessment based on attack trees". In: *Secure IT Systems*. Springer, 2014, pp. 199–212.

[Mai+04] N.A.M. Maiden et al. "Model-driven requirements engineering: synchronising models in an air traffic management case study". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3084 (2004). cited By 21, pp. 368–383.

[Mat+05] S. Mathew et al. "Real-time multistage attack awareness through enhanced intrusion alert clustering". In: *Military Communications Conference, 2005. MILCOM 2005. IEEE*. 2005, 1801–1806 Vol. 3.

[Mat+08] S. Mathew et al. "Insider abuse comprehension through capability acquisition graphs". In: *Information Fusion, 2008 11th International Conference on*. 2008, pp. 1–8.

[MF01] G.B. Magklaras and S.M. Furnell. "Insider Threat Prediction Tool: Evaluating the probability of IT misuse". In: *Computers & Security* 21.1 (2001), pp. 62–73.

[MKM09] Z. Mohaghegh, R. Kazemi, and A. Mosleh. "Incorporating organizational factors into Probabilistic Risk Assessment (PRA) of complex socio-technical systems: A hybrid technique formalization". In: *Reliability Engineering and System Safety* 94.5 (2009). cited By 95, pp. 1000–1018.

[Moh10] Z. Mohaghegh. "Combining system dynamics and Bayesian belief networks for socio-technical risk analysis". In: *ISI 2010 - 2010 IEEE International Conference on Intelligence and Security Informatics: Public Safety and Security* (2010), pp. 196–201.

[MS03] Kevin Mitnick and William Simon. *The Art of Deception: Controlling the Human Element of Security*. Wiley, 2003.

[MT92] K. Meinke and J. V. Tucker. "Universal Algebra". In: *Handbook of Logic for Computer Science*. Ed. by S. Abramsky, D. Gabbay, and T. Maibaum. Vol. I: Mathematical Structures. Oxford University Press, 1992, pp. 189–411.

[NFP98] Rocco de Nicola, Gian Luigi Ferrari, and Rosario Pugliese. "KLAIM: A Kernel Language for Agents Interaction and Mobility". In: *IEEE Trans. Softw. Eng.* 24.5 (1998), pp. 315–330.

[Nid+15] Michael Nidd et al. "Chapter 22 - Tool-based risk assessment of cloud infrastructures as socio-technical systems". In: *The Cloud Security Ecosystem*. Ed. by Ryan KoKim-Kwang Raymond Choo. Boston: Syngress, 2015, pp. 495–517.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Vol. 2283. LNCS. Springer-Verlag, 2002.

[Nur+14] Jason R. C. Nurse et al. "Understanding Insider Threat: A Framework for Characterising Attacks". In: *WRIT'14*. 2014.

[Pau14] Stéphane Paul. "Towards Automating the Construction & Maintenance of Attack Trees: a Feasibility Study". In: *Proceedings First International Workshop on Graphical Models for Security, GraMSec 2014, Grenoble, France, April 12, 2014*. 2014, pp. 31–46.

[PAV14] Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek. "Towards synthesis of attack trees for supporting computer-aided risk analysis". In: *Software Engineering and Formal Methods*. Springer, 2014, pp. 363–375.

[PAV15] Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek. "ATSyRa: An Integrated Environment for Synthesizing Attack Trees". In: *Second International Workshop on Graphical Models for Security (GraMSec'15) co-located with CSF'15*. 2015.

[PDP13a] W Pieters, T Dimkov, and D Pavlovic. "Security Policy Alignment: A Formal Approach". In: *IEEE Systems Journal* 7.2 (June 2013), pp. 275–287.

[PDP13b] W. Pieters, T. Dimkov, and D. Pavlovic. "Security policy alignment: A formal approach". In: *IEEE Systems Journal* 7.2 (2013). cited By 4, pp. 275–287.

[Per14] D Permata Sari. "Attacker Profiling in Quantitative Security Assessment". In: (2014).

[PH08] Christian W. Probst and René Rydhof Hansen. "An extensible analysable system model". In: *Inf. Secur. Tech. Rep.* 13.4 (2008), pp. 235–246.

[PH09a] Christian W. Probst and Jeffrey Hunker. "The Risk of Risk Analysis and its relation to the Economics of Insider Threats". In: *Proc. of the 8th Workshop on the Economics of Information Security (WEIS'09)*. 2009.

[PH09b] C.W. Probst and R.R. Hansen. "Analysing Access Control Specifications". In: *Systematic Approaches to Digital Forensic Engineering, 2009. SADFE '09. Fourth International IEEE Workshop on*. 2009, pp. 22–33.

[PHN07a] Christian W. Probst, René Rydhof Hansen, and Flemming Nielson. "Where Can an Insider Attack?" In: *Formal Aspects in Security and Trust*. Ed. by Theo Dimitrakos et al. Vol. 4691. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 127–142.

[PHN07b] Christian W. Probst, René Rydhof Hansen, and Flemming Nielson. "Where can an insider attack?" In: *Proceedings of the 4th international conference on Formal aspects in security and trust*. FAST'06. Hamilton, Ontario, Canada: Springer-Verlag, 2007, pp. 127–142.

[Pie11a] W. Pieters. "Representing Humans in System Security Models: An Actor-Network Approach". In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 2.1 (2011), pp. 75–92.

[Pie11b] W Pieters. "Representing humans in system security models: An actor-network approach". In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 2.1 (2011), pp. 75–92.

[PWP15] C. W. Probst, J. Willemson, and W. Pieters. "The Attack Navigator". In: *Proceedings of the Second International Workshop on Graphical Models for Security*. Verona, Italy: IEEE, 2015.

[Rob65] John Alan Robinson. "A Machine-Oriented Logic Based on the Resolution Principle". In: *J. ACM* 12.1 (Jan. 1965), pp. 23–41.

[Ros10] Matt Rosenquist. "Prioritizing Information Security Risks with Threat Agent Risk Assessment". In: *Intel White Paper* (2010).

[Sam+13] Layal Samarji et al. "Situation Calculus and Graph Based Defensive Modeling of Simultaneous Attacks". In: *Cyberspace Safety and Security*. Springer, 2013, pp. 132–150.

[Sam11] K.C. Sameer. "Attack Gerenation From System Models". MA thesis. Finland: Aalto University, 2011.

[Sar+14] Anandarup Sarkar et al. "Insider Attack Identification and Prevention Using a Declarative Approach". In: *Security and Privacy Workshops (SPW), 2014 IEEE*. IEEE. 2014, pp. 265–276.

[SBM03] David Scott, Alastair Beresford, and Alan Mycroft. "Spatial Policies for Sentient Mobile Applications". In: *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. Conference Paper. June 2003, pp. 147–157.

[Sch99] Bruce Schneier. "Attack Trees: Modeling Security Threats". In: *Dr. Dobb's Journal of Software Tools* 24.12 (1999), pp. 21–29.

[Sco04] David Scott. "Abstracting Application-Level Security Policy for Ubiquitous Computing". PhD Thesis. PhD thesis. University of Cambridge, Dec. 2004.

[SEH12] T. Sommestad, M. Ekstedt, and H. Holm. "The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures". In: *Systems Journal, IEEE* PP.99 (2012), pp. 1–1.

[SEJ10] Teodor Sommestad, Mathias Ekstedt, and Pontus Johnson. "A probabilistic relational model for security risk analysis". In: *Computers & Security* 29.6 (2010), pp. 659–679.

[SF93] William H Sanders and Roberto S Freire. "Efficient simulation of hierarchical stochastic activity network models". In: *Discrete Event Dynamic Systems* 3.2-3 (1993), pp. 271–299.

[Sha+10] Hamid Reza Shahriari et al. "Vulnerability analysis of networks to detect multiphase attacks using the actor-based language Rebeca". In: *Computers & Electrical Engineering* 36.5 (Sept. 2010), pp. 874–885.

[Sho08] Adam Shostack. "Experiences threat modeling at microsoft". In: *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK*. 2008.

[SPR99] Eric D. Shaw, Jerrold M. Post, and Keven G. Ruby. *Inside the mind of the insider*. Security Management. Dec. 1999.

[Ste91] Daniel F Sterne. "On the Buzzword "Security Policy"". In: *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*. IEEE. 1991, pp. 219–230.

[SYE10] N. Santhi, G. Yan, and S. Eidenbenz. "Cybersim: Geographic, temporal, and organizational dynamics of malware propagation". In: *Proceedings - Winter Simulation Conference* (2010), pp. 2876–2887.

[Thy13] Niels Thykier. "A Reusable Framework for Analysing System Models". MA thesis. DTU Compute, Technical University of Denmark, 2013.

[TML10] Chee-Wooi Ten, Govindarasu Manimaran, and Chen-Ching Liu. "Cybersecurity for Critical Infrastructures: Attack and Defense Modeling". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 40.4 (July 2010), pp. 853–865.

[TREsPASS] The TREsPASS Consortium. *Project web page*. Available at http://www.trespass-project.eu. Accessed January 09th, 2016.

[VNN14] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. "Automated Generation of Attack Trees". In: *Proceedings of the 27th Computer Security Foundations Symposium (CSF)*. IEEE, 2014, pp. 337–350.

[VPH12] Khanh Viet, Brajendra Panda, and Yi Hu. "Detecting collaborative insider attacks in information systems". In: *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, Oct. 2012, pp. 502–507.

[VVM12] Zdenek Vintr, David Valis, and Jindrich Malach. "Attack tree-based evaluation of physical protection systems vulnerability". In: *2012 IEEE International Carnahan Conference on Security Technology (ICCST)*. IEEE, Oct. 2012, pp. 59–65.

[Web78] Max Weber. "Die protestantische Ethik und der Geist des Kapitalismus". In: *Max Weber, Gesammelte Aufsätze zur Religionssoziologie*. 7. edition, (first edition 1920). Tübingen, 1978.

[XAC13] OASIS XACML. "Core Specification: eXtensible Access Control Markup Language (XACML)". In: *OASIS XACML* (2013).

[Xie+09] Anming Xie et al. "A New Method to Generate Attack Graphs". In: *2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement*. IEEE, July 2009, pp. 401–406.

[Zha+11] F Zhao et al. "A hybrid ranking approach to estimate vulnerability for dynamic attacks". In: *Computers and Mathematics with Applications* 62.12 (2011), pp. 4308–4321.